

**MAKİNE ÖĞRENİMİ TABANLI ANDROİD KOD
KARIŞTIRICI TESPİTİ**

Sadık SARIBIYIK



T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNE ÖĞRENİMİ TABANLI ANDROİD KOD KARIŞTIRICI TESPİTİ

Sadık SARIBIYIK
0000-0002-9140-6588

Dr. Öğr. Üyesi Cengiz TOĞAY
(Danışman)

YÜKSEK LİSANS TEZİ
SİBER GÜVENLİK ANABİLİM DALI

BURSA – 2021
Her Hakkı Saklıdır.

TEZ ONAYI

Sadık SARIBIYIK tarafından hazırlanan “Makine Öğrenimi Tabanlı Android Kod Karıştırıcı Tespiti” adlı tez çalışması aşağıdaki jüri tarafından oy birliği ile Bursa Uludağ Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı’nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman : Dr. Öğr. Üyesi Cengiz Toğay

Başkan : Dr. Öğretim Üyesi Cengiz Toğay
: 0000-0001-5739-1784
Bursa Uludağ Üniversitesi,
Mühendislik Fakültesi,
Siber Güvenlik Anabilim Dalı

İmza

Üye : Doç Dr. Metin Bilgin
: 0000-0002-4216-0542
Bursa Uludağ Üniversitesi,
Mühendislik Fakültesi,
Bilgisayar Yazılımı Anabilim Dalı

İmza

Üye : Dr. Öğr. Üyesi Alev Mutlu
: 0000-0003-0547-0653
Kocaeli Üniversitesi,
Mühendislik Fakültesi,
Bilgisayar Yazılımı Anabilim Dalı

İmza

Yukarıdaki sonucu onaylarım

Prof. Dr. Hüseyin Aksel EREN
Enstitü Müdürü

../../....

B.U.Ü. Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmada;

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

.../.../2022
Sadık SARIBIYIK

TEZ YAYINLANMA FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezin/raporun tamamını veya herhangi bir kısmını, basılı (kâğıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma izni Bursa Uludağ Üniversitesi'ne aittir. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet hakları ile tezin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları tarafımıza ait olacaktır. Tezde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanılması zorunlu metinlerin yazılı izin alınarak kullandığını ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederiz.

Yükseköğretim Kurulu tarafından yayınlanan “**Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge**” kapsamında, yönerge tarafından belirtilen kısıtlamalar olmadığı takdirde tezin YÖK Ulusal Tez Merkezi / B.U.Ü. Kütüphanesi Açık Erişim Sistemi ve üye olunan diğer veri tabanlarının (Proquest veri tabanı gibi) erişimine açılması uygundur.

Dr. Öğr. Üyesi Cengiz Toğay

Sadık SARIBIYIK

İmza

Bu bölüme kişinin kendi el yazısı ile okudum anladım yazmalı ve imzalanmalıdır.

İmza

Bu bölüme kişinin kendi el yazısı ile okudum anladım yazmalı ve imzalanmalıdır.

ÖZET

Yüksek Lisans Tezi

Makine Öğrenimi Tabanlı Android Kod Karıştırıcı Tespiti

Sadık SARIBIYIK

Bursa Uludağ Üniversitesi
Fen Bilimleri Enstitüsü
Siber Güvenlik Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Cengiz TOĞAY

Android uygulamalarının kaynak kodları decompiler (Dexdump, Dex2jar, Androguard ve Apktool) adı verilen uygulamalar aracılığı ile elde edilebilmektedir. Tersine mühendislik olarak adlandırılan bu yöntem kötüçül yazılımların tespit edilmesi, kaynak kodlarda yer alan fikri mülkiyet haklarına konu olan bir yöntem ya da içeriğin elde edilmesi, kaynak kod içerisinde bulunan bir bilgi aracılığı ile saldırı yapılması ve kodların değiştirilerek kullanılması gibi birçok senaryoda karşımıza çıkmaktadır. Uygulama geliştiriciler bu tip problemlere karşı uygulamada yer alan sınıf ve değişken isimlerinin karıştırılması, açık metinlerin şifrelenmesi, gereksiz kod ekleme vb. yöntemleri içeren karıştırma tekniklerini kullanmaktadırlar. Bu yaklaşım temelde doğru amaçlara hizmet etmekle birlikte aynı zamanda kötüçül yazılım geliştiriciler tarafından da yoğun olarak kullanılmaktadır. Bu kapsamda bu tip yazılımların tespitinde ilk adım karıştırmanın yapılıp yapılmadığının tespitidir. Tezimizde kod karıştırma tekniklerinden hangilerinin kullanıldığının otomatik ve hızlı bir şekilde belirlenmesine yönelik çalışmalar gerçekleştirilmiştir. Tespitler yapılırken makine öğrenimi tekniklerinden yararlanılmış ve çeşitli algoritma yaklaşımları ile test edilmiştir. IR tekniği için ortalama %94,24. SE tekniği için ise ortalama %92,69 oranında doğruluklarda tespitler yapılmıştır.

Anahtar Kelimeler: Makine Öğrenmesi, Kod Karıştırma Teknikleri, Bilgi Güvenliği
2021, vii + 54 sayfa.

ABSTRACT

MSc Thesis

Machine Learning Based Android Code Obfuscation Detection

Sadık SARIBIYIK

Bursa Uludağ University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Assist. Prof. Cengiz TOĞAY

Source codes of Android applications can be obtained through applications called decompilers (Dexdump, Dex2jar, Androguard and Apktool). This method, which is called reverse engineering, appears in many scenarios such as detecting malicious software, obtaining a method or content that is the subject of intellectual rights in the source code, attacking through information in the source code, and using the codes by changing them. Application developers can avoid such problems by mixing class and variable names in the application, encryption of clear texts, adding unnecessary code, etc. They use mixing techniques including methods. While this approach basically serves the right purposes, it is also heavily used by malware developers. In this context, the first step in the detection of this type of software is to determine whether mixing is done or not. In our thesis, studies have been carried out to automatically and quickly determine which code mixing techniques are used. While making the determinations, machine learning techniques were used and tested with various algorithm approaches 94.24% for the IR technique. For the SE technique, determinations were made with an accuracy of 92.69%.

Key words: Machine Learning , Obfuscation Techniques, Information Security
2021, vii + 54 pages.

TEŞEKKÜR

Bu tezi yazarken hiçbir zaman desteğini esirgemeyen, tecrübelerini paylaşmaktan asla geri durmayan bana bu süreçte yol gösteren çok kıymetli danışman hocam Dr. Öğr. Üyesi Cengiz TOĞAY'a sonsuz teşekkürlerimi sunarım.

Beni bu günlere getiren ve maddi manevi desteklerini hiçbir zaman esirgemeyen kıymetli annem Aynur Sarıbiyık'a kız kardeşlerim Sevban Sarıbiyık, Fatma Ender'e ve sevgili ağabeyim Dr. Öğr. Üyesi Oğuz Yunus SARIBIYIK'a ne kadar teşekkür etsem azdır.

Sadık SARIBIYIK
15/04/2022

İÇİNDEKİLER

ÖZET.....	i
ABSTRACT	ii
TEŞEKKÜR.....	iii
SİMGELER ve KISALTMALAR DİZİNİ	v
ŞEKİLLER DİZİNİ.....	vi
ÇİZELGELER DİZİNİ	vii
1.GİRİŞ	1
2. KURAMSAL TEMELLER ve KAYNAK ARAŞTIRMASI	3
2.1 Kod Gizleme/Karıştırma	3
2.2 Tersine mühendislik araçları	4
2.3 Karıştırıcılar/Gizleyiciler	5
2.4 Gizleme/Karıştırma Teknikleri	8
2.4.1 Tanımlayıcı Yeniden Adlandırma (Identifier Renaming).....	9
2.4.2 Dizi Şifreleme (String Encryption)	9
2.4.3 Kontrol Akışı Karıştırma (Control Flow Obfuscation).....	10
2.5. Makine Öğrenimi Yaklaşımları.....	12
2.6 Androdet	14
3. MATERYAL ve YÖNTEM.....	17
3.1 Veri Seti Oluşturulması.....	17
3.2 Tersine Mühendislik İşlemleri ve Etiketleme	17
3.3 Özellik Çıkarma İşlemi	20
4.BULGULAR.....	29
5. TARTIŞMA ve SONUÇ	52
KAYNAKLAR	54
ÖZGEÇMİŞ	56

SİMGELER ve KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
AES	Advanced Encryption Standard
APK	Android Package Kit
ARM	Acorn RISC Machine
CFG	Control Flow Graph (Kontrol Akış Grafi)
DES	Data Encryption Standart
DEX	Dalvik Executables
GŞS	Gelişmiş Şifreleme Standardı
MIPS	Microprocessor without Interlocked Pipeline Stages
RSA	Rivest Shamir Adleman Algoritması
IR	Tanımlayıcı Yeniden Adlandırma (Identificate Renaming)
SE	Dizi Şifreleme (String Ecrption)
IOS	İphone Operating System(İphone İşletim Sistemi)
DSM	Veri Akış Yönetimi(Data Stream Management)

ŞEKİLLER DİZİNİ

Sayfa

Şekil 2. 1. Obfuscate gizleyicisinin çalışma Mekanizması.....	6
Şekil 2. 2. Dizi Şifreleme Adımları.....	9
Şekil 2. 3. Dizi Şifreleme yapılmış basit bir sözde kod parçacığı.....	10
Şekil 3. 1. APK'nın tersine mühendislik ile kaynak kodlarına ulaşılması.....	18
Şekil 3. 2. Karıştırma yapılmış APK Dosyası.....	19
Şekil 3. 3. Karıştırılmamış APK Özellikleri (IR).....	22
Şekil 3. 4. Karıştırılmamış APK Özellikleri (SE).....	23
Şekil 3. 5. Karıştırılmış APK Özellikleri (IR).....	25
Şekil 3. 6. Karıştırılmamış APK Özellikleri (SE).....	27
Şekil 4. 1 Normal ve Karıştırılmış APK Kaynak Kodu Toplam Alan Sayıları	30
Şekil 4. 2. Normal ve Karışık APK'ların Tek Karakterli Alan Adı Sayıları	30
Şekil 4. 3. Normal APK'ların Tek Karakterli Alan Adları Grafiği.....	31
Şekil 4. 4 Karışık APK'ların Tek Karakterli Alan Adları Grafiği	31
Şekil 4. 5. Aynı APK Dosyasının Şifrelenmiş ve Şifrelenmemiş Halinin Değerleri	32
Şekil 4. 6. Normal ve Karışık APK'ların İki Karakterli Alan Adı Sayıları	32
Şekil 4. 7. Üç Karaktere Sahip Alan Adı Sayısı Grafiği.....	33
Şekil 4. 8. Alan Adları Kelime Boyutu Ortalaması (karıştırılmış)	34
Şekil 4. 9 Alan Adları Kelime Boyutu Ortalaması (Normal).....	34
Şekil 4. 10 Alan Adları Mesafe Ortalaması (Karıştırılmış)	35
Şekil 4. 11. Normal APK'ların Alanlar Arasındaki ort. ASCII mesafesi.	35
Şekil 4. 12. (Karıştırılmış) Ortalama Metot Kelime Sayısı.....	36
Şekil 4. 13 (Normal) Ortalama Metot Kelime Sayısı.....	36
Şekil 4. 14. (Karıştırılmış) Ortalama Metotlar Arası Mesafe.....	37
Şekil 4.15. Tek Karakterli Metot Sayısı.....	37
Şekil 4. 16. Çift Karakterli Metot Sayısı Grafiği	38
Şekil 4. 17. Üç Karakterli Metot Sayısı Grafiği.....	39
Şekil 4. 18. Ort. Sınıf Kelime Boyutu Grafiği	40
Şekil 4. 19. APK'ların Ardışık Sınıflar Arasındaki Ort. ASCII Mesafesi	40
Şekil 4. 20. Normal ve Karışık APK'ların Sınıf Uzunluğunun Özellikleri	41
Şekil 4. 21. İki Karakterli Sınıf Sayısı Grafiği.....	41
Şekil 4. 22. Üç Karakterli Sınıf Sayısı Grafiği	42
Şekil 4. 23. Sınıf Belirlenmesi(Etiketleme) Grafiği.....	43
Şekil 4. 24. APK'ların Dizilerinin Ortalama Karmaşıklık Değerleri.....	44
Şekil 4. 25. Normal ve Karışık APK'ların Dizilerinin Ort. Kelime Boyutları.....	45
Şekil 4. 26. Normal ve Karışık APK'ların Ortalama Uzunlukları	46
Şekil 4. 27. Normal ve Karışık APK'ların Eşittir Simgesi Ortalamaları	47
Şekil 4. 28. Normal ve Karışık APK'ların Ortalama Tire (-) Değerleri	47
Şekil 4. 29. Normal ve Karışık APK'ların Ortalama Artı (+) Sayısı	48
Şekil 4. 30. Normal ve Karışık APK'ların Tekrar Eden Karakterlerin Dağılımı.....	49
Şekil 4. 31. Normal ve Karışık APK'ların Class Parametresi Dağılımı	49

ÇİZELGELER DİZİNİ

Çizelge 3. 1. Özellik çıkarma parametreleri.....	21
Çizelge 3. 2. Yeniden adlandırma yapılan örnek sınıf isimleri.....	24
Çizelge 3. 3. Weka Yazılımı Üzerinden Kullanılan Makine Öğrenimi Algoritmaları. ..	27
Çizelge 4. 1. Makine Öğrenimi Algoritmaları ile modelleme sonuçları.....	50

1.GİRİŞ

İnternete bağlı cihazların sayılarının artmasına bağlı olarak siber güvenlik kapsamında yapılan atak sayılarında da gözle görülür bir artış meydana gelmiştir. Mobil uygulamalar aracılığı ile donanımdan bağımsız geliştirilen siber saldırı uygulamaları aşağıda ifade edilen pek çok senaryoyu gerçekleştirebilmektedir. Tezimizde konu edindiğimiz mobil uygulamalar kapsamında yapılan ataklara örnek olarak;

- Uygulamadan elde edilecek servis IP adresi, kullanıcı adı, şifresi, token, vb. bilgiler ile gerçekleştirilecek ataklar,
- Sunucu bağlantılı uygulamalarda mobil cihazlara güvenen yaklaşımı baz almış geliştiricilerin uygulamada bıraktıkları bilgiler ile bir başka kullanıcının yetkilerine sahip olmak,
- Uygulamanın bir benzerini Android mağazasına yükleyerek kullanıcıların hesap bilgilerinin çalınması,
- Mağazada ifade edilen amacı dışında yeteneklere (ortam dinleme, resim çekme, telefon rehberi bilgileri, SMS çalma, SMS atma vb.) sahip uygulamaların geliştirilmesi verilebilir.

Android işletim sistemi için geliştirilen uygulamalar doğrudan Android mağazalarında yer bulabilmektedir. Kaynak kodları değerlendirilmediği için kötücül yazılım barındıran uygulamalar mağazada yerini alabilmektedir. Bu tip yazılımların tespitine yönelik uygulanabilecek yaklaşımların başında statik analizler yer almaktadır. Statik analizlerde, anti-virüs uygulamaları tarafından da yoğun bir şekilde kullanılan, kötücül kod parçasına ait bir imzanın aranması yöntemine başvurulmaktadır (Chen et al., 2013). Kötücül yazılımları hazırlayan geliştiriciler bu yöntemleri aşmak için kodlarını karıştırmakta ve tespit edilmesini zorlaştırmaya çalışmaktadırlar. Dolayısı ile karıştırılmış bir yazılımın kötücül bir yazılım olması ihtimali bulunmaktadır.

Siber atakların gerçekleştirilmesinde istemcilerin sağladığı bilgiler kullanılabilir. Normal şartlarda istemci tarafından yapılan bir kontrol (örnek olarak "kullanıcı" adı en fazla 10 karakter olur) bir python uygulaması ile aynı servise 11 karakter gönderilerek test edilebilir, aşırı yükleme gibi yaklaşımlar ile sunucu çalışmaz hale getirilebilir.

Normal şartlar altında istemciye güvenen yaklaşıma sahip uygulama geliştirilmemesi gerekmektedir.(Faruki, Hossein, Vİjay, Mauro, & Manoj, 2017) Ancak, geliştiricilerin zaman baskısı, deneyim eksikliği ve sadece uygulamada fonksiyon ekseninde düşünmeye bağlı üretilen uygulamalarda yukarıda bahsedilen basit hatalar gerçekleştirilebilmektedir. Bu tip açıkların gizlenmesi ve sunucuda yer alan servisler, servisler ile olan iletişime ait parametre gibi bilgilerin gizlenmesi için karıştırma teknikleri kullanılabilir. Bazı uygulamalar barındırdıkları yöntem, parametre gibi çalınması durumunda üreticisini zarara sokacak içerik barındırıyor olabilirler. Bu tip bilgi ve içeriğinde karıştırılarak sunulmasında fayda vardır.

Yukarıda ki bilgiler ile değerlendirildiğinde her karıştırılmış yazılım kötücül yazılım olarak değerlendirilmesi doğru değildir (Zhu et al., 2020). Kod karıştırma işlemini yapmak için kullanılabilecek hali hazırda ticari araçlar (Allatori , Dasho, Proguard ve Dexguard) mevcuttur. Ancak unutulmamalıdır ki, karıştırma işlemleri sadece tersine mühendislik ile kod parçasından elde edilecek kazanımın elde edilmesinin zorlaştırmasını hedeflemektedir. Karıştırma işlemi sırasında isimlendirme, bazı içeriğin şifrelenmesi, boş kod parçalarının serpiştirilmesi vb. yaklaşımlar kodun çözülmesi ya da bir içeriğin elde edilmesi sırasında geçirilecek süreyi uzatmaktadır.

Tezimizde karıştırılmış yazılımlarda en yaygın kullanılan karıştırma tekniklerinden hangisinin kullanıldığının tespitine yönelik bir çatı uygulamasına yer verilmiştir. Makine öğrenmesi tabanlı önerilen yaklaşımda veri seti olarak F-droid ve Google Play Store’da yer alan uygulamalardan yararlanılmıştır.

2. KURAMSAL TEMELLER ve KAYNAK ARAŞTIRMASI

Android işletim sistemi hali hazırda pazar piyasasında açık ara en çok tercih edilen işletim sistemi olarak karşımıza çıkmaktadır.(Chua & Balachandran, 2018) Bir diğer işletim sistemi olan İOS kapalı kalmayı bir şirket politikası olarak kabullendiğinden ötürü dışarıya açık değildir. Bu sebepten açık kaynak kodlu işletim sistemi olan android pazarda tercih edilmektedir.

Android cihazların kişisel bilgisayarların sayısını çoktan aştığı bilinen bir gerçektir. Bu bilgiler ışığında kötücül yazılımcıların android cihazları hedef alması ise olağan bir durum olarak kabul edilebilir. 2014 yılında yapılan bir araştırmaya göre kötücül yazılımların hedeflerinin %99'undan fazlası android işletim sistemini hedef almaktadır. (Maiorca, Ariu, Corona, Aresu, & Giacinto, 2015)

2.1 Kod Gizleme/Karıştırma

Kod gizleme/karıştırma, günümüzde uygulamaların tersine mühendislik yöntemleri ile elde edilebilen kaynak kodlarının açığa çıkarılmasına karşı kullanılan yöntemlerin genel adıdır. Kod gizleme ile Android uygulaması özelinde sınıf adları, kelime boyutları, ardışık seçilen iki tanımlayıcı arası mesafeler ve kodlarda kullanılan özel karakterler farklılığa uğrar ancak uygulamanın yaptığı iş veya kodun çıktısında bir değişim söz konusu olmamaktadır (Bichel B, raychev V 2016). Uygulamaların kaynak kodlarında gizleme/karıştırma yaparken bir takım şekilsel değişiklikler yapıldığı herkes tarafından bilinen bir gerçektir. Bununla birlikte değiştirilen kodların anlamsal yönden bir değişiklik yapılmadığı da bir gerçektir. Bu durum tersine mühendislik araçları ve işlemleri için zorluklar çıkarır. Kodlara ulaşmanın maliyetini arttırır. Ancak yeterli maliyet ve süre harcandığında ulaşılabilir hale gelecektir. Anlamsal bazda ya da kodların yaptığı nihai işlem bakımından değişiklik olmadığından bu durum gerçekleşmektedir. Gizleme tekniklerinin çözilemeyen noktası bu durum olarak nitelendirilebilir. Kodların içeriklerini değiştirme yoluna gidildiğinde çıktılar farklılaşacak ve kaynak kodu niteliğini yitirecektir. Kodların sadece şekilsel bazda değiştirilmesi durumunda ise maliyet artacak ve tersine mühendislik süreçleri zorlaşacaktır ancak yine de imkansız olmayacaktır.

Bahsedilen karıştırma işlemlerini uygulayabilmek adına bir takım tekniklerden yararlanılmaktadır. En çok kullanılan karıştırma teknikleri tanımlayıcı yeniden adlandırma, dizi şifreleme ve kontrol akışı karıştırmadır.

2.2 Tersine mühendislik araçları

Gizleme işlemleri çeşitli karıştırıcılar ve çeşitli tekniklerle yapılmaktadır. Gizleme işlemlerinden bahsetmeden önce APK dosyalarının içeriği hakkında bilgi vermek faydalı olacaktır; Android uygulamaları bir şekilde tersine mühendislik yöntemleri ve araçları ile arşiv halinden kurtarılması durumunda karşımıza aşağıdaki dosyalar ve kaynak kodları çıkmaktadır. Bu dosyalar APK dosyalarının temel yapıtaşlarını oluşturmaktadır.

Lib: Uygulamanın ARM ve MIPS gibi farklı CPU mimarisi için derlenmiş paylaşılan kitaplıklarını içerir. Paylaşılan kitaplıklar işlemciye bağlı olduğundan genellikle C ile yazılırlar.

Assets: Uygulamaya özel ek varlıklara sahiptir. Bunlar, daha sonra bir bayt akışı olarak yüklenebilen görüntüler gibi gerçekten ham dosyalardır.

META-INF: Kullanıcı verilerini içermemesi gereken dahili bir Java meta dizini olarak düşünülebilir. Ortak dosyalar, geliştirici imza dosyasıdır (sertifikaya benzer; .RSA veya .DSA olarak adlandırılır); paketteki tüm dosyaları listeleyen MANIFEST.MF dosyasına bir şifreleme karması dahildir. (Google tarafından Google uygulama mağazasında oluşturulması zorunlu tutulmuştur, ([Google, 2020c,a](#)))

AndroidManifest.xml: İzinler ve donanım gereksinimleri gibi uygulamayla ilgili temel ayrıntıları içerir.

Class.dex: Dalvik Sanal Makinesinde (DVM) veya Android Çalışma Zamanında (ART) çalışan yürütülebilir dosyadır. DEX dosyasının genellikle programın ana bölümünü içerdiğini ve kolayca yeniden derlenebileceğini unutulmamalıdır. Bu nedenle, giderek daha fazla uygulama, önemli kodları paylaşılan kitaplıklara dış kaynak sağlamaya başlamış durumdadır.

Resources.arsc: Önceden derlenmiş kaynakları içerir ve genellikle performans nedenleriyle kullanılır. Bu dosyaya bir nevi log dosyası denilebilir.

Gizleme yapmak için hedeflenen dosyanın dex dosyaları olduğundan bahsedilmektedir (Zhang et al., 2021). APK Dosyalarının değerli kısımlarının neredeyse tamamı .dex dosyası içerisinde bulunmaktadır. Sınıflar, alan adları, yöntem adları, tanımlayıcılar, diziler, algoritmalar gibi kaynak kodları bakımından anlam ifade eden ne varsa .dex dosyasının içerisinde bulunmaktadır.

Gizleme işlemlerini yapabilmek adına APK dosyaları üzerinde bir takım açığa çıkartmalar yapılmalıdır. Bir noktada APK dosyaları JAR(Java Archive) dosya formatının bir türevidir, çünkü APK dosyaları java üzerinde derlenmektedir. Hal böyle olunca bir APK dosyasını olduğu gibi alarak işlem yapmak pek mümkün değildir. Bunun yerine APK dosyalarının içeriğine ulaşmak için birtakım işlemlere uğraması gerekmektedir. APK dosyalarının jar halinden çıkarılması için birtakım çıkarıcılar vardır (V et al., 2015). Bahsi geçen çıkarıcılardan en popüler olanları ise apktool, dex2jar ve soot'tur. Yapılan çalışmalarda Apktool'un başarısı göze çarpmaktadır. Apktool geri birleştirilen hallerinde uygulama işlevlerinin %97sinin tekrar elde edilmesini sağlamıştır.(Arnatovich et al., 2018)

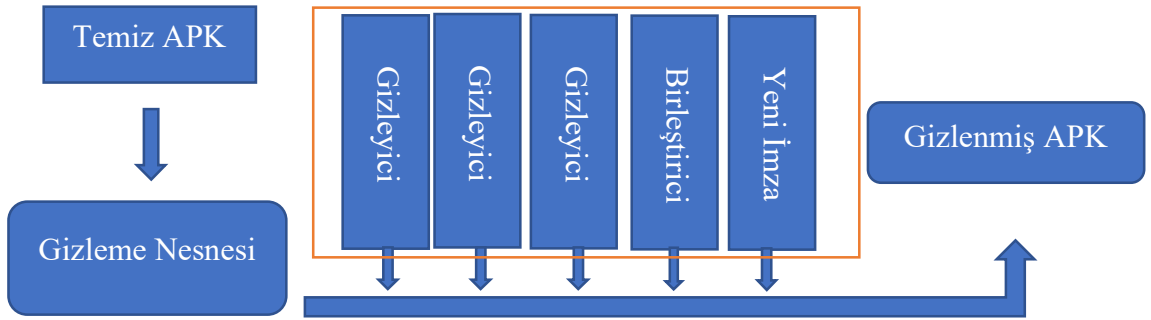
Öte yandan bu 3 çıkarıcı ile kıyası yapılmayan ancak bahsetmeye değer bir diğer çıkarıcı daha mevcuttur. Bahsetmemize neden olan ise tersine mühendislik yapılması esnasında kullanım kolaylığı sağlaması, basit bir yapıya sahip olması ve kaynak kodlarını tam olarak karşımıza çıkarabilmesi olarak sıralanabilir. (<https://github.com/skylot/jadx>)

Uygulamaların kaynak kodlarına eriştikten sonra ise yapılması gereken kod karıştırmak/gizlemektir. Kod karıştırmak için çeşitli uygulamalar vardır. Bir takım ticari faaliyet gösteren firmaların yanı sıra açık kaynak kodlu olarak da yapılabilecek ortamlar mevcuttur.

2.3 Karıştırıcılar/Gizleyiciler

Obfuscapk karıştırıcıların ticari olmayanlarından biridir. Github üzerinden kolaylıkla elde edilip kullanılabilir durumdadır. Obfuscapk, modüler ve genişletilmesi kolay olacak

şekilde tasarlanmıştır, bunun sağlanabilmesi için ise bir eklenti yönetim sistemi olan Yapsy (Python uygulamaları için basit bir eklenti sistemi) üzerine kurulmuştur. Sonuç olarak, her bir karıştırıcı, soyut bir temel sınıftan miras alan ve gizleme yöntemini uygulaması gereken bir eklentidir. Araç, bir APK'yı işlemeye başladığında, gerekli tüm bilgileri (yani, geri derlenmiş kodun konumu) ve işlemlerin dahili durumunu (yani, zaten kullanılmış olan gizleyicilerin listesi) depolamak için bir gizleme nesnesi oluşturur. Ardından, gizleme nesnesi, gizleme yönteminin bir parametresi olarak tüm etkin eklentilere/gizleyicilere sırayla iletilir. Etkin durumda ki eklentilerin listesi ve sırası, komut satırı seçenekleri aracılığıyla belirtilir. Araç, yeni gizleyicilerle kolayca genişletilebilir: src/obfuscapk/obfuscators dizinine şaşırtma tekniğini uygulayan kaynak kodu ve eklenti varlık verilerini (obfuscator-name.obfuscator) eklemek yeterlidir. Araç, başka yapılandırma adımlarına gerek kalmadan yeni eklentiye otomatik olarak algılamaktadır. (Aonzo et al., 2020)



Şekil 2. 1. Obfuscapk gizleyicisinin çalışma Mekanizması

Proguard, özellikle bir Java dosya küçültücü, optimize edici, karartıcıdır (gizleyici, karıştırıcı). Bununla birlikte Proguard, Kotlin ile de çalışır. Kotlin, Mayıs 2019 tarihinde Google tarafından uygulama geliştiricilerin kullandıkları bir dil olduğu duyuruldu. Kotlin'in en belirgin özelliği, kısa ve öz olmasında yatmaktadır. Kotlin hem yazma, test etme hem de bakım açısından daha az kod gerektirir. Bu yönüyle de pek çok uygulama geliştiricisine cazip gelmektedir.

Proguard, kullanılmayan sınıfları, alanları, yöntemleri ve nitelikleri algılayıp kaldırarak Kotlin uygulamasını küçültür. Bir sonraki aşama, yöntemlerin bayt kodunun analiz edildiği ve optimize edildiği yerdir. Kalan sınıflar, alanlar ve yöntemler daha sonra hiçbir anlam ifade etmeyen daha kısa adlarla karıştırılır. Proguard uygulandıktan sonra

uygulama nihayet bir APK dosyasına yerleştirilir ve oluşturulan apk, .zip formatına dönüştürülür ve class.dex dosyası .zip dosyasından çıkarılır. Class.dex dosyası, class.dex'i class.jar'a ayıştıran dex2jar'a verilir. Bu class.jar dosyası jd-gui kullanılarak açılır. Gizlenmiş kaynak kodu artık görüntülenebilir hale getirilmiş durumdadır. Proguard'ın uygulanmasının temel amacı karıştırmanın amacıyla aynıdır ve uygulamanın tersine mühendisliğini ve herhangi bir kötü amaçlı kaynak kodunun eklenmesini çok zor hale getirmektedir.

Öte yandan DexGuard ise ProGuard'ın ticari bir versiyonudur. Ticari sürüm olmasından ötürü bir takım fonksiyonları daha geniş kapsamlıdır ve ücretsiz versiyonuna göre gizleme yaptırılırken daha fazla seçeneğe sahip olunmaktadır. DexGuard, doğrudan Dalvik bayt kodunu hedef alır. Daha gelişmiş karıştırma işlemleri yaptığından ötürü DexGuard ile gizlenmiş uygulamalarının tersine mühendislik yöntemleriyle tekrar elde edilmesine çok zor gözüyle bakılmaktadır. (Reva University et al., n.d.) Bunun nedeni, sınıf ve yöntem adlarının ASCII olmayan karakterlerle değiştirilmesi ve dizelerin şifrelenmesidir. Bilindik yöntemlerde karıştırma işlemleri esnasında sadece harf ve rakamlar kullanılır oysa DexGuard'ta bununla birlikte ASCII tablosunda bulunmayan karakterlerinde sınıf ve değişken isimlerinin yeniden adlandırılmasında kullanılmasıdır. APK Protect, Android yürütülebilir dosyalarına özel başka bir gelişmiş, kullanıma hazır şaşırtma ve koruma hizmetidir. Koruma kabuğu özelliğini bir APK'ya uygulayarak, başka hiç kimsenin bu APK'da hata ayıklama yapabilmesine olanak tanımamaktadır. Fakat ilginçtir ki .dex dosyasında 'APKProtected' dizesi bulunduğundan, kötü amaçlı yazılımdaki kullanımını belirlemek kolaydır. Bu durumda bu yöntem sadece iyi huylu uygulamaların gizlenmesinde etkili olacağı söylenebilir. (Apvrille & Nigam, 2014)

HoseDex2Jar, bilgisayar korsanlarının bir Android uygulamasının kaynak koduna erişmek için tersine mühendislik araçlarından biri olan Dex2Jar kullanmasını engelleyen basit bir araçtır. HoseDex2Jar, class.dex dosyasına Dex2Jar'ın işleyemeyeceği bazı talimatlar ekler. Bu işlem ise uygulamaya dex2jar yapılmasına karşı bir kalkan gibi iş görür ve tersine mühendislik yapılmasına engel olur.

(<https://github.com/strazzere/dehoser/>)

Dasho, tüm Java ve Android uygulama dosyaları olan apk'lar için oldukça üst düzeyde koruma sağlayarak korsanlık, fikri mülkiyet hırsızlığı ve kurcalanma riskini azaltır. Gizleme ile ilgili hizmetler arasında, yöntemlerin, değişkenlerin adlarının yeniden adlandırılması, uygulamanın hassas bölümlerinde dizelerin şifrelenmesi (genelde .dex dosyasında olmaktadır) ve kullanılmayanları bulmak için kodun statik analizine izin veren budamayla optimizasyon yer alır. (<https://www.preemptive.com/products/dasho/>)

2.4 Gizleme/Karıştırma Teknikleri

Gizleme/karıştırma yapmak için birtakım yöntemler geliştirilmiştir. Bu bölümde onlardan bahsedilecektir. Gizleme teknikleri iki amaç için kullanılabilir. İyi huylu uygulamalar tersine mühendislik işlemlerinden etkilenmemek, güvenlik açığı vermemek ve fikri mülkiyetlerini korumak. Kötücül uygulamalar ise tespit edilmelerini engellemek için gizleme tekniklerinden faydalanmaktadır.

Gizleme tekniklerinin asıl üç adet ana kullanım sebebi vardır. Bunlardan en önemlisi tabi ki koruma, kötü amaçlı davranışları maskeleye (kötücül uygulamalar için) ve son olarak da kod optimizasyonudur.

Korumanın hedefi uygulamaların tersine mühendislik yapılabilme ihtimalini azaltmak mümkünse imkansız hale getirmektir. Fikri mülkiyet haklarının korunması ise uygulama geliştiricilerin maddi ve manevi kaynakları ile oluşturdukları eserlerinin izinsiz şekilde bir başkasının eline geçmesinin önüne geçmek içindir. Uygulamanın kendi çıkarları için kullanımının önüne geçmek bir nevi yukarıda bahsedilen fikri mülkiyet haklarının ihlali ile benzerlik gösterse de burada asıl bahsedilmesi gereken nokta uygulamanın şeklen ikizinin üretilerek kullanıcıların aldatılmasıdır. Uygulamanın görünürde aynı uygulama olup arka planda kötücüle dönüştürülmesi.

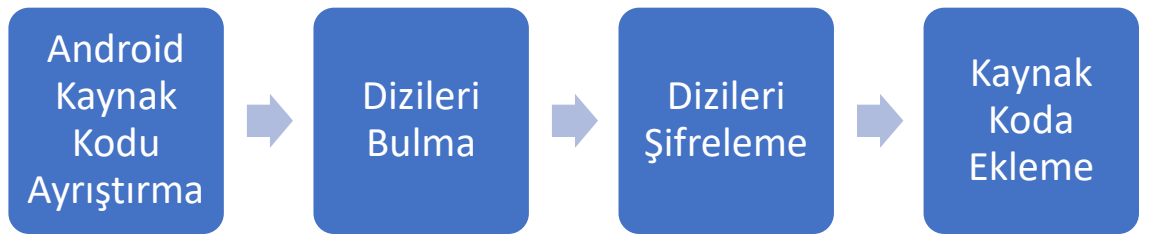
Kötücül davranış maskeleye, kötü amaçlı yazılım inceleyen sistemlerin radarlarından kaçınabilmeyi amaçlar. Uygulamaları iyi huylu kötücül diye sınıflandırmak için kullanılan bazı parametreleri değiştirerek karar verme mekanizması bozulmaya çalışılabilir. Bir diğer noktası ise uygulama imzasının iyi huyluymuşçasına değiştirilerek virüs programı gibi tarayıcılardan kaçınmasını hedefler. (Callberg et al, n.d.)

2.4.1 Tanımlayıcı Yeniden Adlandırma (Identifier Renaming)

Bu yöntemde ifadeleri daha az anlamlı ve genelde harf ya da rakamla değiştirilmek suretiyle gerçekleştirilir. En popüler yöntemlerden biridir. Değiştirme işlemi sonrasında kodda görüntüsel anlamda farklar meydana gelecektir ancak bu durum kodun çıktısında hiçbir değişikliğe sebep olmamaktadır. Bu yöntemin bir diğer adı ise tanımlayıcı yeniden adlandırmadır (Identifier Renaming : IR).

Değişkenler, sınıflar yöntemler gibi kod kısımları kodun okunabilirliği açısından yani yazılımcılar açısından önemlidir. Oysa ki tanımlayıcıların isimlerinin ne olduğu programı ilgilendiren ya da sonuçlarını etkileyen bir durum değildir. Bu yüzden de her tanımlayıcı istenildiği şekle ya da harfe sayıya hatta ASCII tablosunda bulunan herhangi bir karaktere dönüştürülebilir (Dong et al., 2018).

2.4.2 Dizi Şifreleme (String Encryption)



Şekil 2. 2. Dizi Şifreleme Adımları.

Dizeler yazılım üretim aşamalarında oldukça sık kullanılan veri yapılarının başında gelir. Dize Gizleme, hassas bilgileri ortaya çıkaran düz metin dizelerini gizler. Hassas düz metin bilgileri kötüye kullanılabilir. Dizeler, çalışma zamanında düz metin biçiminde mevcut olmalıdır. Bu nedenle, geliştirici düz metin dizesini şifrelemeyi tercih etmek isteyebilir. Dizi Gizleme, AES, DES veya XOR şifrelemesi kullanan tersine çevrilebilir bir şifreleme işleviyle kullanılabilir. Sonuçta ortaya çıkan, orijinal düz metnin yerini alır. Dinamik analiz, çalışma zamanı bilgilerini çıkardığı için kullanışlıdır. Şifrelenmiş kod, okunabilir dizi bilgilerini ASCII karakterli bir dizi diziye dönüştürür ve

kodun okunabilirliğini gizler (Li et al., 2017). Aşağıdaki şekil 2.2’ de bu gizleme tekniği kullanılmış bir örnek gösterilmektedir.

Deneme_dizi= “merhaba dünya”;	Deneme_dizi = yeni dizi(a.a(“asfJfaHaTjy=”));
-------------------------------	---

Şekil 2. 3.Dizi Şifreleme yapılmış basit bir sözde kod parçacığı.

Şekil 2.3’de basitçe gösterildiği gibi “merhaba dünya” yazılı bir dizi parçası örneğini “a.a” şeklinde kısa ve tersine mühendislik yapan birisi tarafından tamamen anlamsız hale getirilmiştir.

2.4.3 Kontrol Akışı Karıştırma (Control Flow Obfuscation)

Oldukça popüler ve popüler olduğu kadar diğerlerine nispeten bir miktar daha karmaşık bir gizleme tekniği olan kontrol akışı güçlendirme tekniğinde, asıl amaçlanan şey kodun olağan akışı kesilerek analistin kodu anlayabilmesinin önüne geçmeyi hedefler. Kontrol akışı karartması, bir uygulamanın kontrol akışını dönüştürmeyi amaçlar, böylece dönüştürülmüş kontrol akışı statik olarak zorlayıcı ve çok sayıda olası temel blok sıralama kombinasyonuna sahip olur, bu durum uygulamanın anlaşılmasını engeller. Genel olarak, kod akışını analiz etmeyi çok daha karmaşık hale getirmek için bir uygulamanın kontrol akışını bozmanın iki yolu vardır. Bunlardan ilki, kod bloklarının yazıldığı doğrusal sıralamayı bozmaktır. Bir diğeri, yöntem içindeki kontrol akışı dallarını veya sapmalarını arttırmaktır. Bu ise aralara ölü kodlar ekleyerek yapılmaktadır. (Balachandran et al., 2016) Bu tekniği uygularken bazı alt teknikler kullanılır. Bunlardan kısaca bahsedilmesi gerekirse:

CallIndirection: Bu teknik, kod semantiğini etkilemeden kontrol akış grafiğini değiştirir; orijinal olanları çağıran yeni yöntemler ekler. Örneğin, ‘a’ yöntemine yapılan bir çağrı, orijinal yöntem a’yı çağıran yeni bir çağrı yöntemi b ile değiştirilecektir.(Bacci, Bartoli, Martinelli, Medvet, & Mercaldo, 2018)

Goto: Bir metot verildiğinde, metodun sonuna işaret eden bir goto komutu ve ilk gotodan sonra talimata işaret eden başka bir goto ekler; iki yeni düğüm ekleyerek CFG'yi değiştirir. Bu durum ise kodun tersine mühendislik yapılmasına negatif etki eder.

Yeniden Sıralama: Bu teknik, koddaki temel blokların sırasını değiştirmeyi içerir. Bir dal talimatı bulunduğunda, koşul tersine çevrilir ve hedef temel bloklar buna göre yeniden sıralanır. Ayrıca, git talimatlarını kötüye kullanan kodu rastgele yeniden düzenler.

Aritmetik Dal: Bu, koda bazı işe yaramaz ve anlamsal koruyucu talimatlar eklemeyi amaçlayan bir tekniktir. Bu durumda, önemsiz kod, aritmetik hesaplamalardan oluşur ve bu hesaplamaların sonucuna bağlı olarak bir dal talimatı, dal asla alınmayacak şekilde hazırlanır.

Noop: No-operasyonun kısaltması, hiçbir şey yapmayan özel bir talimattır. Bu teknik, her yöntem uygulamasına rastgele noop talimatları (yani önemsiz kod) ekler.

Yukarıda belirtilen gizleme tekniklerinin kullanıldığı durumlar için yapılan bazı çalışmalar gizleme yapılan ve karıştırılmış hale gelen uygulamaların tekrardan aslına döndürülmesine yönelik bir takım çalışmalar gerçekleştirmektedir. Bu mekanizmaya TIRO (Target-Instrument-Run-Observe) adı verilmiştir. Dil tabanlı ve çalışma zamanı tabanlı yapılan gizlemeyi otomatik olarak algılayıp tersine çevirebileceği vaat edilmektedir. (Wong & Lie, n.d.)

Öte yandan sınıf düzeyinde şaşırtma tekniği tanınması için geliştirilen bir başka yöntemde ise kaynak kodları üzerinden paragraf vektörleri denen etiketleme yöntemleri vasıtası ile elde edilen veriler ışığında makine öğrenimi algoritmaları üzerinden kullanılan gizleme tekniğinin tespitinde çalışmalar yapılmıştır. (Park, You, Cho, Park, & Han, 2019)

Yapılan çalışmalarda kontrol akışı gizleme tekniğinin hem karıştırma işlemlerinde hem de işlemin tamamlanması aşamalarında oldukça sorunla karşılaşmıştır. Özellikle tersine mühendislik yapılması aşamasında uygulamaların çözülemediği bu sebeple de karıştırma işleminin tamamlanamadığı görülmüştür. Tüm bu durumlar ışığında kontrol akışı gizleme tekniğinin analizden çıkarılmasının daha iyi bir çözüm olacağından o yönde adım atılmıştır.

2.5. Makine Öğrenimi Yaklaşımları

Makine Öğrenimi insanların öğrenme şekillerini taklit ederek veri ve algoritma kullanımına odaklanıp doğruluğunu da kademeli olarak arttıran yapay zeka ve bilgisayar bilimi dalıdır.

Bir problemin akış şeması şeklinde doğrudan veya kolayca açıklayamadığımız, fakat çözüme yönelik veri kümesi oluşturulabilen durumlarda böylesi bir öğrenmeye ihtiyacımız olabilmektedir. Makine öğreniminin veri madenciliğinden farkı ise makine öğrenimi öğrendikleri karşısında davranışını değiştirir. (Lee, Taylor, & Kalpathy-Cramer, 2017)

Makine Öğrenimi yöntemlerinde pek çok algoritma bulunmaktadır. Bunlar veri madenciliği, görüntü işleme, tahmine dayalı analiz vb. gibi birden fazla alanda faal olarak kullanılır haldedir. Belirli bir işlemci gücüne sahip ve kalıcı bir şekilde programlanmamış makineler, verilerin üzerinde ki bazı desenleri veya kalıpları tespit ederek ileride daha iyi seçimler yapmak üzere tahminlerde bulunmak için verilerle öğrenme sürecinde ilerlerler. Burada asıl hedeflenen nokta insan müdahalesi ya da yardımı olmadan kendi kendilerine öğrenmeleri ve tahminler geliştirmeleridir.

Denetimli öğrenme olarak adlandırılabilen bir takım algoritmalar bulunmaktadır.

- Karar Ağacı; seçimleri ve sonuçları tıpkı bir ağaç dallanması gibi makineleştiren bir algoritmadır. Her ağaç düğümlerden ve dallardan oluşmaktadır. Her düğüm sınıflandırılması hedeflenen grubu temsil eder ve her dal ise düğümlerin alabileceği değerleri temsil edebilmektedir.
- Navie Bayes; Tahmin ediciler arasında bağımsızlık varsayımı ile bayes teoremine dayalı bir sınıflandırma tekniğidir. Naive bayes teoremi ise kısaca bahsedecek olunur ise, rassal değişken için olasılık dağılımı içinde koşullu olasılıklar ile marjinal olasılıklar arasındaki ilişkiyi gösterir. (Mahesh, 2018)

Destek vektör Makinesi; Makine öğreniminde destek vektör makineleri, sınıflandırma analiz regresyon analizi için kullanılan denetimli öğrenme algoritmasıdır. Girdilere

özellik uzayında eşleyerek doğrusal olmayan bir sınıflandırmayı verimli şekilde gerçekleştirebilir.

- Regresyon; iki teori için kullanılabilen bir tekniktir, bunlardan ilki tahmindir bir diğeri ise bağımsız ve bağımlı değişkenler arasında nedensel ilişkileri belirlemek için regresyon analizi kullanılabilir.
- Model ağaçları tekniği (M5 Tree); Verilerin yapısal bir temsilini ve sınıfın parçalı doğrusal bir uyumunu sağlar. Geleneksel ağaç yapısına sahiptirler ancak yapraklarda ayrık sınıf etiketleri yerine doğrusal fonksiyonlar kullanırlar. Geleneksel karar ağacı öğrencilerine benzer şekilde M5, verileri tahmine dayalı niteliklerin değerlerine göre bölerek bir ağaç oluşturur. (Biabani, Halaghi, & Ghorbani, 2016)
- RepTree; RepTree algoritması hızlı bir karar ağacı öğrencisidir. Bilgi kazancını/varyansını kullanarak bir karar/regresyon ağacı oluşturur ve bunu hatası azaltılmış budama (geri düzenlemeli olarak) kullanarak budama yapar. Sayısal değerleri sadece bir kez sıralama yapar, bu durum karmaşıklık açısından olumlu bir etki olarak görülebilir. (Ridzuan Daud & Wolfe Corne, n.d.)
- Bagging (Bootstrap Aggregating); İnsanlar bir karar alacağı zaman tek bir yerden ya da sadece kendileri karar almazlar ki doğrusu da birden çok yerden tartışarak konu hakkında bilgiler toplamaktır. İnsan davranışlarını taklit ederek daha otomatize şekilde işlemleri yapmayı hedefleyen makine öğrenimi algoritmalarında da benzer durum ele alınır. Bu duruma ise torbalama (Bagging) denilmektedir. Bu işlem yapılırken geçmiş sonuçların ortalaması alınarak bir tahminde bulunulur. Torbalama verilen bir eğitim setini kullanıp simüle ederek makine öğrenimi yöntemlerinin kararsız kalmasının önüne geçmeye çalışır. (Tu, Shin, & Shin, 2009)
- Rastgele Komite (Random Comittee): bu yaklaşım, bir dizi ana sınıflandırıcı üretir ve olasılık tahminlerini oluşturur. Her ana sınıflandırıcı benzer verilere dayanır ancak başka bir rastgele gen sayısı kullanır. Yalnızca ana sınıflandırıcı

rastgele seçildiğinde anlamlı hale gelir, aksi takdirde tüm sınıflandırıcılar eşdeğer olabilmektedir.

- Karar Tahtası (Decision Stump): Karar tahtası tek seviyeli bir karar ağacıdır. Örnekleri özellik değerlerine göre sıralayarak sınıflandırır. Bir karar tahtasında, her düğüm, sınıflandırılacak bir örnekteki bir özelliği temsil eder ve her dal, düğümün alabileceği bir değeri temsil eder. Örnekler, kök düğümünden başlayarak sınıflandırılır ve özellik değerlerine göre sıralanır. En kötü ihtimalle, bir karar kütüğü en sağduyulu temel çizgiyi yeniden üretecektir ve seçilen özellik özellikle bilgilendirici ise daha iyi sonuç verebilmektedir. Güçlendirme, bir tür güçlü topluluk yöntemidir. Hızlandırma, verileri yeniden örnekleyerek bir sınıflandırıcılar topluluğu oluşturur ve ardından çoğunluk oylama stratejisini kullanarak sınıflandırıcıları birleştirir. Her bir sınıflandırıcıya beslenen eğitim verilerinin dağılımını, özellikle her eğitim örneğinin ağırlıklarını açıkça değiştirir. Başlangıçta, ağırlıklar tüm eğitim örnekleri için aynıdır. Yükseltme prosedürü sırasında, her sınıflandırıcının eğitimi tamamlandıktan sonra ayarlanırlar. Yanlış sınıflandırılan numuneler için ağırlıklar artırılırken, doğru sınıflandırılan numuneler için ağırlıklar azaltılır. Nihai grup, bireysel sınıflandırıcıların kendi doğruluklarına göre birleştirilmesiyle oluşturulur. (Duan, Dong, Xi, & Ma, 2018)

2.6 Androdet

Androdet, 2 farklı veri seti üzerinde çalıştırılarak elde edilen değerler üzerinden online bir şekilde modelleme yaparak daha yüksek doğruluk oranı elde etmeyi amaçlayan bir yapıda kurgulanmıştır. 3 farklı karıştırma tekniğini tespit etmeye çalışmaktadır. Bahsi geçen gizleme teknikleri ise tanımlayıcı yeniden adlandırma, dizi şifreleme ve kontrol akış gizlemesi olarak adlandırılmaktadır. Bu karıştırma teknikleri mobil pazarda en popüler olan gizleme teknikleri olarak bilinmektedir ve uygulanmaktadır. Ancak kontrol akış gizlemesi birkaç ticari gizleyici yazılım tarafından sunulsa da yaygınlığı ve tespiti henüz yapılmamıştır. Yapılan çalışmada her bir gizleme tekniği için ayrı ayrı bir modül bulunmakta olup eğitimleri dahil pek çok durumda ayrı ayrı hizmet verebilmektedir. Androdet, DSM olarak bilinen ve Türkçesi “Veri Akış Yönetimi” olan bir teknikle çalışmaktadır. DSM, geleneksel veri madenciliği algoritmaları tarafından kullanılan statik

yöntemlerin aksine modellerini giderek büyüten bir yöntemdir. Bu nedenle de, geleneksel veri madenciliğine dayalı tüm makine öğrenmesi yöntemleri toplu öğrenme algoritmaları olarak bilinir ve veri akışı madenciliğini kullananlara çevrimiçi öğrenme algoritmaları denilebilir. Bu noktada geleneksel makine öğrenimi yöntemleri tüm veri ile eğitilerek oluşturulur ancak çevrimiçi öğrenme de bölüm olarak ilerler, bazı noktalar belirlenir ve o noktaya kadar elde edilen veriler artık sistemin doğruluk oranıdır ve eski veriler atılır. Tüm diğer yapay zeka destekli sistemlerde olduğu gibi veri kümesi arttıkça ve sistem veri ile beslendikçe daha efektif sonuçlar elde etmeye imkan tanımaktadır.

Öte yandan sınıflandırmak için 3 stratejiyi benimsemektedir. İlk olarak en iyi kombinasyon setini bulabilmek için tüm hiperparametre değerleri kombinasyonları açgözlü bir şekilde dener, ikinci durumda tüm kombinasyonları tekrar ancak bu kez rastgele bir şekilde araştırır. Bunun avantajı ise çözümü aç gözlü (greedy) aramasından daha hızlı şekilde bulabilmesidir. Üçüncü yol ise her ne kadar en uygun kombinasyon kümesini garanti etmese de sınırlı sayıda denemeye rastgele bir arama kullanmaktır.

Hoeffding Tree, Weighted Majority Algorithm, Leveraging Bag, LearnNSE, Stokastik Gradient Descent (SGD) ve Naive Bayes de dahil olmak üzere MOA tarafından sağlanan çok çeşitli algoritmalar kullanılıp sonuçlar elde edilmeye çalışılmıştır.

Naive Bayes sınıflandırıcısı; örüntü tanıma problemine ilk bakışta oldukça dar bir önerme ile kullanılabilen olasılıksal bir yaklaşımdır. Örüntü tanımada kullanılan parametrenin istatistiksel açıdan bağımsız olmasını gerekli kılar.

Stochastic Gradient Descent (SGD); Stochastic Gradient Descent (SGD), Destek Vektör Makineleri(SVM) ve Lojistik Regresyon gibi dışsal kayıp fonksiyonları altında lineer sınıflandırıcıların ayırt edici öğretiminde basit ama çok etkili bir yaklaşımdır. SGD, makine öğrenmesi alanında uzun zamandır vardır ancak son zamanlarda büyük ölçekli öğrenme bağlamında dikkat çektiği bilinmektedir.

Bagging(Torbalama); Torbalama, güçlendirme ve Rastgele Ormanlar, tek sınıflıların performansını iyileştirmek için kullanılan klasik topluluk yöntemleridir. Tek sınıflandırıcıların doğruluğunu ve çeşitliliğini artırarak üstün performans elde ederler.

Çoğunluklu Ağırlık Algoritması (Weighted majority algorithm-WMA); Herhangi bir tür öğrenme algoritması, sınıflandırıcı ve hatta gerçek insan uzmanları olabilen bir tahmin algoritmaları havuzundan bir bileşik algoritma oluşturmak için kullanılan bir meta öğrenme algoritmasıdır . Algoritma, havuzdaki algoritmaların doğruluğu hakkında önceden bilgi sahibi olmadığını varsayıyor, ancak bir veya daha fazlasının iyi performans göstereceğine dair tahminler içeriyor. Bileşik algoritmayı oluşturmak için havuzdaki algoritmaların her birine pozitif bir ağırlık verilir. Bileşik algoritma daha sonra havuzdaki tüm algoritmalarından ağırlıklı oylar toplar ve daha yüksek oyu olan tahmini verir.

Hoeffding Ağacı Algoritması: akış veri sınıflandırmasında bir karar ağacı öğrenme yöntemi olarak bilinmektedir. İlk meydana çıktığı zamanlarda web analiz ve tahminleri üzerinde kullanılan bir algoritmadır. Tipik bir geleneksel toplu öğrencilerinkine neredeyse benzer karar ağacını üretmektedir.

Androdet tarafından kullanılan başlıca algoritmalar bu şekilde sıralanabilir. Bunun haricinde otomatik ayarlanmış modeller ve destek vektör makinesi de kullanılmıştır.

3. MATERİYAL ve YÖNTEM

Bu bölümde veri setleri, kullanılan algoritmalar, yazılımlar, yöntemler ve araçlardan bahsedilmiş olup çalışmaya konu olan yapının oluşturulması aşamaları hakkında bilgiler içermektedir.

3.1 Veri Seti Oluşturulması

Tez çalışmasında gizlenmiş/karıştırılmış uygulamaları tespit edebilmek adına öncelikle veri seti oluşturulması çalışmalarına başlanılmıştır. Veri seti oluşturulması amacıyla pek çok araştırma da yararlanılan APK deposu F-Droid'ten faydalanılmıştır. Toplu şekilde APK dosyalarını elde edebilmek uzun süreç alacağından bu konuda toplu indirme yöntemleri araştırılmış ve Wget platformunun işlevselliği ön plana çıkmıştır.

Wget kısaca anlatmak gerekirse bir internet sitesinde bulunan materyalleri elde etmenize olanak tanıyan bir yazılım paketidir. İndirme işlemi esnasında ise yaygın olarak kullanılan http, https, ftp, ftps protokolleri üzerinden paketleri elde etmenize olanak tanır. Komut satırı üzerinden etkileşimsiz bir şekilde yazılımın komutları yazılarak indirme yapılabilir.

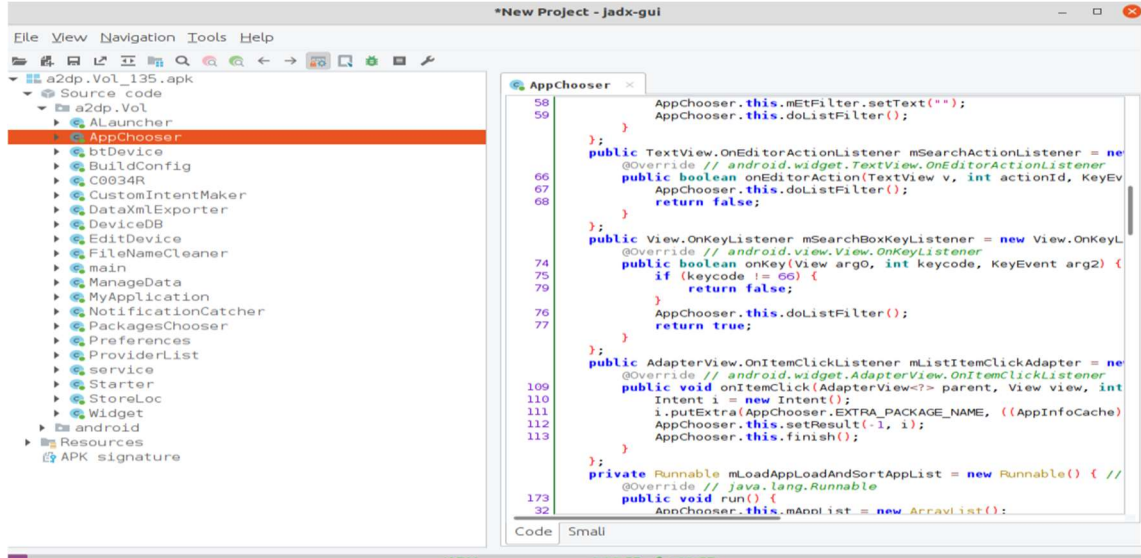
Bu işlemlerin yapılması neticesinde f-droid sitesinde bulunan 2800 adet apk dosyası kısa sürede indirilmiştir. Veri seti indirildikten sonra ise etiketleme işlemlerine geçilmiştir. Burada etiketleme işleminden kast edilen ise indirilen APK dosyalarında gizleme/karıştırma işleminin yapıp yapılmadığının tespit edilmesidir.

3.2 Tersine Mühendislik İşlemleri ve Etiketleme

Kullanıcılar tarafından kullanılan yazılımlarla tersine mühendislik yaparak APK dosyalarının kaynak kodu gibi temel yapılarına ulaşma imkanı tanımaktadır.

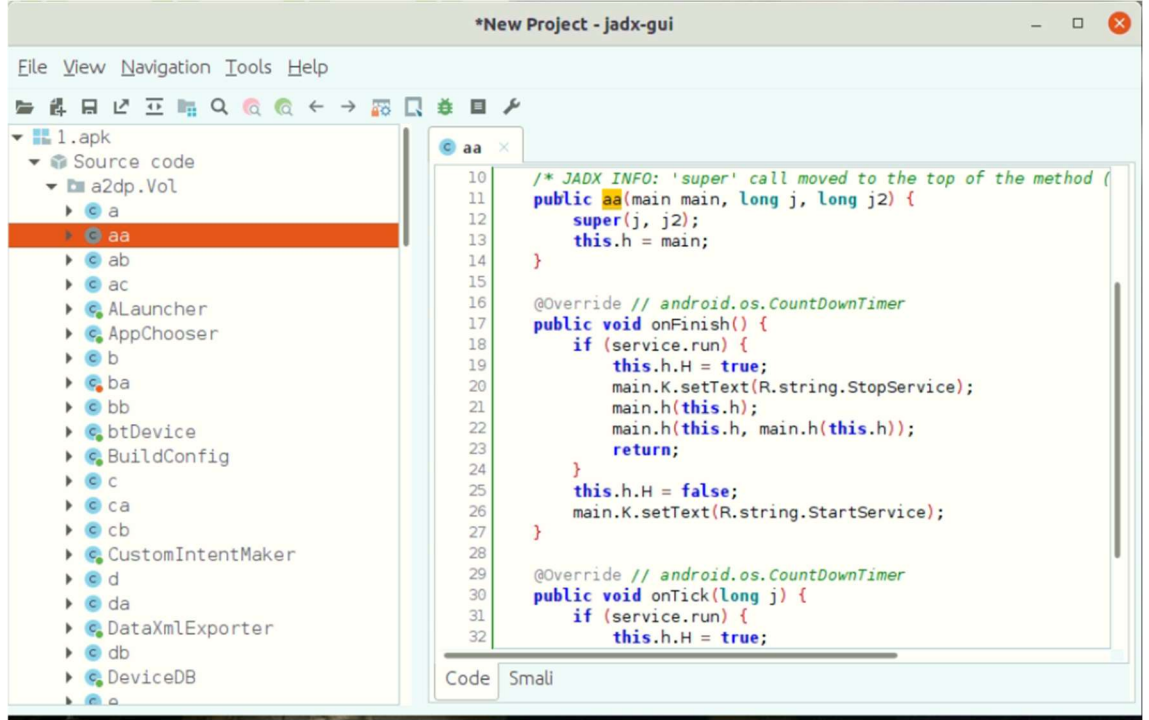
Kontrol etmek istediğimiz APK dosyasını Jadx-Gui ile tersine mühendislik işlemine maruz bırakarak kaynak kodlarına erişmeye başlayabiliriz. Tersine mühendislik aşamasının ardından kaynak kodlarına ulaşıldığından artık APK dosyasının karıştırılıp karıştırılmadığı kontrol edilebilir hale gelmiş bulunmaktadır. Şekil 3.2'de görüldüğü

üzere sorunsuz bir şekilde tersine mühendislik işlemi yapılmıştır. Kaynak kodlarından anlaşıldığı üzere herhangi bir karıştırma tekniği kullanılarak karıştırma/gizleme yapılmadığı anlaşılmaktadır.



Şekil 3. 1. APK'nın tersine mühendislik ile kaynak kodlarına ulaşılması.

Şekil 3.2'de gizleme/karıştırma yapılmamış uygulamanın kaynak kodları gözükmeyle birlikte üzerinde çalışılacak olan detektör için de asıl kullanılacak olan veri seti elemanlarının bir örneği görülmektedir. Nitekim gizleme yapılmayan uygulamaların olması analizler için daha uygun olacaktır. Detektörün doğru çalıştığının ölçülebilmesi için öncelikle tespit etmeye çalıştığımız gizleme tekniğinin kesin bir şekilde uygulandığından emin olmamız gerekmektedir. Dolayısıyla elimizde hangi gizleme tekniği ile gizlendiği bilinmeyen bir uygulama varsa ve detektör bunu tespit etmişse bu tespitin doğruluğundan emin olunamayabilir. Oysa ki hangi yazılım tarafından karıştırıldığı bilinen APK dosyalarının doğru analiz edilip edilmediği ve detektörün doğru çalışıp çalışmadığı net bir şekilde gözlemlenebilmektedir. Karıştırma yapılmış uygulamalar ilk etapta özellikleri çıkartılırken kontrol grubu olması açısından oldukça önemlidir. Karıştırma yapılmamış uygulama özellikleri baz değer olarak elde tutulup karıştırılma yapılmış uygulamaları tespit etmek açısından modellemelerde karşılaştırma yapılmasına imkan vermektedir.



Şekil 3. 2. Karıştırma yapılmış APK Dosyası

Resimde görüldüğü üzere tanımlayıcıların isimleri harf ve rakam kombinasyonları ile yeniden adlandırılmıştır. Böylece tersine mühendislik yapıldığı halde hangi sınıfın ne işe yaradığı hakkında bilgi sahibi olunamamıştır.

İlk etapta F-Droid adlı APK deposundan elde edilmiş bulunan 2800 adet APK dosyası yukarıdaki işlemler ile teker teker incelenmiştir. Yapılan bu ayıklama ve etiketleme işlemlerinin ardından ise F-Droid deposundan alınan APK dosyalarından 2192 tanesinin gizleme/karıştırılma yapılmamış uygulamalardan oluştuğu tespit edilmiştir. Geriye kalan 608 adet APK dosyasında ise tanımlayıcı yeniden adlandırma, dizi şifreleme gibi gizleme teknikleri kullanılarak karıştırma işlemi yapıldığı gözlemlenmiştir.

Yapılan etiketleme ve ayrıştırma işlemlerinin ardından sırada tespit edilen APK dosyalarının özelliklerini çıkarma işlemine geçilmiştir.

3.3 Özellik Çıkarma İşlemi

Özellik çıkarma işlemleri sırasında bu alanda daha önce çalışma yapılmış olan Androdet adı verilen ve çevrimiçi öğrenme tekniklerinden beslenerek sürekli çalışmayı vadeden bir mekanizmanın özellik çıkarma yazılımından faydalanılmıştır. Ancak bu yazılım hazır halde olmadığından ve deneysel bir çalışma olduğundan hazır hale getirilmesi için birtakım düzenlemeler ve geliştirmeler yapılması gerekmiştir. Öte yandan yayınlarında belirtildiği üzere kontrol akış gizlemesi tekniği için kullanılması planlanan tersine mühendislik araçları ve özellik çıkarıcı kısımlarında sorunlarla karşılaşmıştır. Nitekim karşılaşılan sorunlar bir noktadan itibaren sürecin ilerlemesini sekteye uğratar hale geldiğinden kontrol akış gizlemesi tekniği araştırmadan çıkarılmıştır. Geriye kalan iki karıştırma tekniğinde ise mekanizmanın çıkardığı özellikler aşağıdaki tabloda listelenmiştir. (Mirzaei et al., 2019)

Veri setimizi doğru bir sınıflandırmanın ardından teker teker dosyaların kaynak kodlarına ulaşarak yani tersine mühendislik yapmak suretiyle işlemler tamamlanmıştır. Özellik çıkarma işleminin neticesinde veriler .arff uzantılı veri dosyası içeriğini yazarak işlem tamamlanmıştır. Androdet mekanizması Çizelge 3.1’de gösterilen parametrelerde kaynak kodundan bir takım değerler elde etmiştir.

Çizelge 3. 1. Özellik çıkarma parametreleri.

Tanımlayıcı Yeniden Adlandırma	Dizi Şifreleme
Avg_Wordsize_Flds	Avg_Entropy
Avg_Distances_Flds	Avg_Wordsize
Num_Flds_L1	Avg_Length
Num_Flds_L2	Avg_Num_Equals
Num_Flds_L3	Avg_Num_Dashes
Avg_Wordsize_Mtds	Avg_Num_Slashes
Avg_Distances_Mtds	Avg_Num_Pluses
Num_Mtds_L1	Avg_Sum_RepChars
Num_Mtds_L2	
Num_Mtds_L3	
Avg_Wordsize_Cls	
Avg_Distances_Cls	
Num_Cls_L1	
Num_Cls_L2	
Num_Cls_L3	

Öncelikle özellikler temiz (karıştırma yapılmamış) APK dosyaları üzerinden çıkarılmıştır. Temiz APK dosyalarının özellik çıkarma işlemi tamamlandıktan sonra ise .arff dosyası şeklinde elimize geçen dosyalarda ki veriler aşağıdaki şekilde gösterilmiştir. Burada gösterilen veriler virgülle ayrılmış olup her virgül arasında ki değerler yukarıdaki tabloda belirtilen parametrelerin kaynak kodlarındaki değerlerin hesaplanması ile elde edilmiştir. Bu hesaplama ve ölçüm işlemlerini yapan yazılım ise python dilinde yazılmıştır.

```

@relation Android Apps Features for IR detection
@attribute Avg_Wordsize_Flds real
@attribute Avg_Distances_Flds real
@attribute Num_Flds_L1 integer
@attribute Num_Flds_L2 integer
@attribute Num_Flds_L3 integer
@attribute Avg_Wordsize_Mtds real
@attribute Avg_Distances_Mtds real
@attribute Num_Mtds_L1 integer
@attribute Num_Mtds_L2 integer
@attribute Num_Mtds_L3 integer
@attribute Avg_Wordsize_Cls real
@attribute Avg_Distances_Cls real
@attribute Num_Cls_L1 integer
@attribute Num_Cls_L2 integer
@attribute Num_Cls_L3 integer
@attribute class integer
@data
67.9571,994.3665,13,59,101,65.4863,722.1673,3,10,28,73.9397,1195.0686,1,1,2,1
62.9625,699.8495,24,80,142,64.714,677.8536,3,6,27,73.506,1290.5075,1,1,8,1
67.7079,960.2096,5,22,41,65.8982,722.552,3,3,16,76.1937,1352.0203,1,1,1,1
58.669,522.2139,5,2,54,59.4952,409.9,0,0,3,57.1111,412.2353,1,0,0,1
67.5182,990.4518,15,42,108,65.6721,729.4209,14,15,43,72.8839,1186.5628,1,1,5,1
68.45,1011.9986,4,9,20,65.191,700.0195,3,2,9,78.8533,1438.2741,1,0,0,1
67.3958,942.6979,5,33,51,65.7942,723.423,3,4,17,76.6979,1351.4154,1,1,1,1
50.0,3.5,9,0,0,57.84,540.5417,6,0,1,60.8,590.2105,2,0,4,1
67.5823,983.6638,9,27,87,65.6513,741.5142,12,14,41,76.3995,1388.4248,1,1,4,1
66.9371,870.5147,0,0,6,63.3293,629.6304,2,1,4,74.9945,1368.2274,1,0,1,1
67.3455,937.8029,10,38,82,65.5501,732.037,12,19,51,72.0261,1140.9474,1,1,6,1
67.0077,946.2053,17,44,106,66.2617,768.794,10,6,27,73.3698,1205.2828,1,1,3,1
68.6552,1051.3643,14,44,92,66.2557,741.653,3,12,27,74.7771,1263.5125,1,1,0,1
69.3342,1081.0194,8,31,67,66.3757,741.1794,3,5,21,73.4324,1171.9439,1,1,2,1
67.7163,845.6811,2,2,1,63.408,613.5671,0,0,0,68.1449,861.2628,1,0,0,1
67.4013,967.0375,16,11,24,64.808,680.9794,2,2,6,77.4628,1420.8456,1,0,0,1
61.6014,673.1507,2,3,15,61.5359,636.1937,0,4,13,66.4246,708.4183,1,0,1,1
68.0278,977.4666,4,10,22,65.1374,696.4303,3,2,9,78.1408,1383.6398,1,0,0,1
68.2446,1027.3096,13,47,114,66.0505,756.6804,12,18,51,72.7402,1156.3725,1,1,5,1
68.877,1064.9617,10,19,28,65.5356,697.104,3,3,13,73.5095,1199.2002,1,1,0,1
69.6006,1119.6868,5,27,60,66.2903,746.2393,5,6,16,72.0397,1129.0625,1,0,2,1
63.5513,724.2573,12,21,50,64.7102,682.8668,4,1,9,67.4059,902.6551,1,0,3,1

```

Şekil 3. 3. Karıştırılmamış APK Özellikleri (IR)

Şekil 3.3'te tanımlayıcı yeniden adlandırma gizleme tekniğine göre incelenen APK'ların matematiksel olarak özelliklerini barındıran dosyanın içeriği görülmektedir.

Şekil 3.4'te görülen veriler her bir virgül arasında en üstte belirtilen özellik bakımından APK dosyasının kaynak kodlarının özellikleri olarak görülmektedir. Bu veri tipi .arff dosya yapısının bilinen yapısıdır. Oluşturulan sayısal veriler, kaynak kodların metinsel ifadelerinin söz dizimsel olarak sayısallaştırılmış halidir. Bir APK dosyasının içerisinde ki kaynak kodlarında kaç adet alan adı mevcut sınıf adlarının tek karakterli olanlarının sayısı gibi veriler içermektedir.


```

@relation String Features
@attribute Avg_Entropy real
@attribute Avg_Wordsize real
@attribute Avg_Length real
@attribute Avg_Num_Equals real
@attribute Avg_Num_Dashes real
@attribute Avg_Num_Slashes real
@attribute Avg_Num_Plus real
@attribute Avg_Sum_Freq_Chars_h1 real
@attribute class integer
@data
3.3145,101.9571,39.2272,0.0887,0.497,0.0202,0.0192,31.4517,1
2.9158,67.278,18.1823,0.0816,0.0886,0.0392,0.0047,11.5128,1
3.2972,82.1066,29.2537,0.1167,0.3133,0.0815,0.0065,21.6003,1
2.6043,59.2424,9.3515,0.0,0.0066,0.0306,0.0022,3.286,1
2.9953,76.1062,23.9225,0.08,0.2219,0.0207,0.0139,17.0187,1
3.373,74.1041,25.1041,0.1262,0.0349,0.0107,0.0,17.216,1
3.3244,81.7067,29.4081,0.1201,0.3001,0.0538,0.0072,21.7203,1
2.6088,62.8333,13.8333,0.0517,0.0862,0.2874,0.0172,7.8218,1
3.3205,79.179,27.6284,0.0866,0.1882,0.0182,0.01,20.0025,1
3.237,71.3816,22.3816,0.2035,0.0652,0.0668,0.0079,14.8919,1
3.2298,75.6107,24.7369,0.0808,0.1975,0.0377,0.0121,17.2202,1
3.1305,74.5381,23.7654,0.0683,0.2017,0.0208,0.0101,16.5299,1
2.8641,74.9197,22.6207,0.0733,0.1867,0.0137,0.0163,16.0896,1
3.3438,83.3849,32.0419,0.1147,0.2126,0.0152,0.015,24.3686,1
1.1261,63.5007,3.3431,0.0069,0.0023,0.017,0.0005,0.898,1
3.2746,72.2389,23.1929,0.1159,0.0381,0.0195,0.0009,15.6053,1
2.53,61.523,12.523,0.0443,0.1005,0.0716,0.0239,6.7922,1
3.366,74.2646,25.2646,0.1287,0.1009,0.0175,0.0,17.3713,1
3.3113,77.0984,26.3308,0.0745,0.1437,0.0202,0.0082,18.7409,1
3.2314,84.632,30.1134,0.096,0.4182,0.0185,0.0224,22.6429,1
3.2732,74.7833,25.4164,0.0876,0.0526,0.059,0.0033,17.9291,1
3.1252,67.4578,18.4578,0.1296,0.049,0.0748,0.0027,10.8811,1
2.7552,66.5278,15.7986,0.0251,0.1299,0.0273,0.0061,9.5927,1
1.2435,64.1711,6.2754,0.0403,0.0714,0.0157,0.0421,3.1253,1
3.083,69.0191,19.9809,0.1978,0.0542,0.2201,0.0048,13.0542,1
3.3152,80.4185,28.7648,0.075,0.2255,0.0417,0.0126,21.1324,1
2.9774,63.0,14.0,0.0,0.0,0.0,0.0,6.5824,1
3.109,70.5398,21.5398,0.1344,0.0432,0.0583,0.0,14.5082,1
3.2444,78.269,26.4962,0.0703,0.2137,0.0158,0.0121,19.1209,1

```

Şekil 3. 4. Karıştırılmamış APK Özellikleri (SE)

Temiz APK dosyalarının özelliklerini elde ettikten sonra karıştırılmış uygulamaların özelliklerinin elde edilmesi gerekmektedir. Bunun için daha önceleri de bahsedilen kontrol grubu oluşturmaya ihtiyaç vardır. Bu sebepten daha önce özellikleri çıkarılan 2192 adet APK dosyasının hepsinin karıştırılarak karıştırılmış halinin de özelliklerinin çıkarılmasına ihtiyaç duyulmuştur.

Bu karıştırma işlemini yapabilmek adına piyasada bulunan karıştırıcılar araştırılmış ticari ve açık kaynak kodlu karıştırıcılar arasından Allatori'nin kullanılması uygun görülmüştür. Allatori mantık olarak hemen her tanımlayıcı ismini yeniden adlandırabilir

bu durum. gizlemenin/karıştırmanın yanı sıra aynı zamanda kaynak kodunun boyutunu ve optimizasyonunu da sağlar.

Allatori gizleme yaparken gizlemenin derinliğini seçmenize olanak tanır. Varsayılan gizleme ile işlem yapıldığında, tüm sınıflar gizlenmez ama gizleme yapılan paketlerin değerleri varsayılan pakete taşınır. Tam gizleme yapılması durumunda ise kesinlikle tüm sınıfların isimleri yeniden adlandırılır ve yeniden adlandırılan sınıflar da varsayılan paketine taşınır. Burada bahsedilen varsayılan paket bir nevi gizleme logları tutulması anlamını ifade eder. Döngü yapıları, tersine mühendislik araçlarının tanıyamayacağı şekilde parçalara ayrılır. Bu, algoritmaları daha az okunabilir kılmak ve yoğun, sıkı kod parçalarına uzunluk eklemek için bir yaklaşımdır. Ek olarak, dizeler çalışma zamanında gizlenir ve kodu çözülür. Bu, normalde insan tarafından okunabilen ve analistlere iyi önerilerde bulunabilecek mesajları ve adları içerir. (Schulz & Plohmann, n.d.)

Allatori veya diğer karıştırıcılar olsun yeniden adlandırma yaparken harf ve rakam kullanılarak gizleme tercih edilir ancak özel karıştırma yapılması durumunda yeniden adlandırmaların arasına ASCII tablosunda bulunan herhangi bir karakter de eklenebilir.

Çizelge 3. 2. Yeniden adlandırma yapılan örnek sınıf isimleri

Gizleme yapılmadan önceki sınıf ismi	Gizleme yapıldıktan sonra sınıf ismi
Customer_name	Aaa
Birth_date	00ab
Color_of_smthng	Abab

Çizelge 3.2’de tanımlayıcı yeniden adlarına tekniğinin çalışmasına dair kısa bir örnek gösterilmektedir. Allatori çalıştırılıp hali hazırda özelliklerini çıkardığımız APK dosyalarının gizlenmesi için kullanılmaya başlanılmış ve seçtiğimiz iki adet gizleme tekniği ile gizlenmesi gerçekleştirilmiştir. Gizleme/karıştırma işleminin ardından daha evvel karıştırılmamış hallerinin özellikleri çıkarılan APK dosyalarının bu sefer yeniden özellik çıkarması yapılmıştır.


```

@relation Android Apps Features for IR detection
@attribute Avg_Wordsize_Flds real
@attribute Avg_Distances_Flds real
@attribute Num_Flds_L1 integer
@attribute Num_Flds_L2 integer
@attribute Num_Flds_L3 integer
@attribute Avg_Wordsize_Mtds real
@attribute Avg_Distances_Mtds real
@attribute Num_Mtds_L1 integer
@attribute Num_Mtds_L2 integer
@attribute Num_Mtds_L3 integer
@attribute Avg_Wordsize_Cls real
@attribute Avg_Distances_Cls real
@attribute Num_Cls_L1 integer
@attribute Num_Cls_L2 integer
@attribute Num_Cls_L3 integer
@attribute class integer
@data
71.7819,1162.1677,52,49,21,64.3989,691.5326,35,7,27,74.2839,1309.5922,26,25,3,1
72.7847,1206.4164,52,24,5,64.5123,696.4285,29,1,12,78.616,1428.166,5,0,0,1
61.4275,610.8595,52,53,65,61.341,530.8301,11,3,49,64.9768,842.8274,26,23,5,1
58.6963,558.1567,22,0,2,63.3138,670.1592,7,0,3,57.3,801.7848,26,15,1,1
53.6452,387.337,52,8,1,63.3992,720.2061,15,0,4,75.4405,1655.9391,25,38,0,1
71.6559,1188.557,52,39,17,64.5295,698.6794,33,7,22,75.3075,1390.3736,26,85,1,1
52.243,249.0943,52,29,0,62.9608,787.783,52,82,5,59.4775,814.6242,26,39,131,1
61.5687,601.5826,52,61,69,62.0367,575.8976,31,11,56,66.7677,983.0342,26,27,7,1
62.1834,696.6588,30,1,1,61.6334,551.4163,7,1,5,58.0897,725.0452,26,37,2,1
68.6384,1087.1348,52,235,40,66.1439,796.501,36,14,24,70.8977,1186.8295,26,393,2,1
72.4917,1278.5121,52,29,16,64.6636,702.9683,28,1,12,77.1073,1382.9883,18,1,0,1
72.4931,1227.9942,52,53,18,64.7085,697.9683,35,3,24,73.852,1275.8067,26,32,5,1
72.3639,1277.4555,52,67,70,65.2186,720.9554,37,11,42,71.2834,1122.7576,26,28,19,1
60.5581,708.1392,52,22,9,62.2083,649.586,17,4,10,70.4446,1490.7394,26,85,1,1
71.7111,1185.9492,52,23,13,64.2369,686.3047,24,0,7,76.8478,1385.6591,5,0,0,1
56.2,443.9184,10,2,1,60.0,587.0,3,0,0,54.5263,405.0556,6,0,1,1
62.6903,758.9288,52,20,1,62.7934,629.3238,14,0,4,78.1657,1582.5015,5,0,0,1
72.1407,1188.4889,52,26,8,65.0203,700.5162,28,1,9,77.668,1374.3333,7,1,0,1
75.5497,1277.3453,52,58,34,65.5845,744.2397,51,13,36,73.8303,1303.1417,26,45,6,1
71.9587,1171.3241,52,26,8,65.029,699.7735,28,1,9,77.5384,1370.5199,6,0,0,1
71.4689,1221.9283,52,39,41,63.8126,671.6318,33,4,15,71.7715,1201.459,25,81,7,1
61.5341.828.5977.16.0.1.58.7551.524.7708.5.0.1.58.2593.722.7692.7.0.0.1

```

Şekil 3. 5. Karıştırılmış APK Özellikleri (IR)

Şekil 3.5'te Karıştırılmış durumda olan APK'ların matematiksel olarak özelliklerini içeriin dosya görülmektedir. Grafiğe aktarıldığında arada ki farklar daha net olarak gözükecektir.

Tanımlayıcı yeniden adlandırma karıştırma tekniğine dair çıkarılan özellikler .arff dosyasına işlenmektedir. Şekil 3.6'da özellikleri çıkarılmış veri setimizde bulunan APK dosyalarına ait veriler gözlenmektedir.

Her iki durumda da elde edilen .arff dosyalarının teknik analiz ve doğruluk tespiti yapılabilmesi açısından makine öğrenimi yöntemleri ile değerler elde edilmiştir. Bu yöntemlerde test edilirken Weka uygulaması kullanılmasına kararlaştırılmıştır.

Weka yazılımında verilerimizi test etmek için .arff dosya tipinde bulunan verilerimizin .csv veri tipinde olması gerektiği fark edilip python dili kullanılarak dosyaların dönüşümleri gerçekleştirilmiştir. Bu dönüşümün yapılma sebebi weka yazılımının

verileri arff olarak içeriye aktarırken çok daha hassas bir şekilde karşılık vermesidir. Verilerde ki en küçük hata bile weka'nın çalışmasına engel olmuştur. Ancak unutulmamalıdır ki, tersine mühendislik işlemleri ile APK'ların tamamına ulaşılammamaktadır. Ulaşılsa dahi tekrardan eski haline dönüşlerde sorunlar meydana gelmektedir. Tersine mühendislik işlemleri ile ulaşılp tekrar birleştirilse dahi arff dosya tipinde ki her bir özellik parametreleri her bir APK için doldurulamadığından arff olarak incelenmesi ve denemelerin yapılmasına olanak sağlamamıştır.

.arff uzantılı dosya ile uyumlu çalışan weka yazılımında hataların alınma sebebi; weka yazılımının satır sonlarında “,” beklemesinden ötürü dosyanın içerisinde ki verileri okuyamamıştır.

```
@relation String Features
@attribute Avg_Entropy real
@attribute Avg_Wordsize real
@attribute Avg_Length real
@attribute Avg_Num_Equals real
@attribute Avg_Num_Dashes real
@attribute Avg_Num_Slashes real
@attribute Avg_Num_Plus real
@attribute Avg_Sum_Freq_Chars_h1 real
@attribute class integer
@data
3.1249,66.461,17.4099,0.1583,0.1032,0.0956,0.1055,8.7051,1
3.1294,66.6557,17.6557,0.0598,0.0948,0.1663,0.1823,8.9679,1
2.8511,62.7305,13.7305,0.0437,0.084,0.0865,0.0814,5.8367,1
3.2253,67.228,18.228,0.2383,0.044,0.0104,0.0959,8.829,1
2.7596,62.1913,13.1913,0.1531,0.1541,0.06,0.0796,5.5264,1
2.6927,64.8709,14.2772,0.1317,0.1641,0.1164,0.1472,6.9878,1
3.2048,70.1443,20.8671,0.1613,0.1537,0.1426,0.2188,11.5656,1
3.0486,64.4849,15.0681,0.1187,0.1506,0.149,0.1203,6.4291,1
2.7617,65.1333,16.1333,0.0696,0.1623,0.2507,0.1464,8.5333,1
3.3061,67.8809,18.6947,0.1311,0.1356,0.1309,0.158,9.1486,1
3.1045,66.8376,17.8376,0.1613,0.1479,0.1191,0.1284,9.2263,1
3.1867,67.4903,18.2614,0.0977,0.1295,0.0994,0.09,9.2065,1
3.1415,68.9997,19.3881,0.128,0.1378,0.1078,0.0951,10.6513,1
2.5104,59.8677,10.8677,0.0808,0.1162,0.0886,0.1036,4.2012,1
3.1462,67.0533,18.0533,0.0963,0.1531,0.0894,0.1917,9.4015,1
1.7279,54.6341,5.6341,0.0732,0.0,0.0732,0.0244,1.0976,1
3.1564,64.6645,15.6645,0.0212,0.0277,0.0244,0.0114,6.5,1
3.1913,68.9818,19.9147,0.1712,0.1461,0.1558,0.1377,11.0126,1
3.1542,67.0732,17.384,0.1574,0.1094,0.1364,0.1118,8.5197,1
3.1114,66.719,17.7018,0.0987,0.1332,0.1173,0.0842,8.9841,1
2.7763,61.8432,12.8113,0.0972,0.0991,0.1309,0.087,5.3215,1
3.2125,67.9167,18.9167,0.0,0.0,0.0,0.0,9.9583,1
3.1511,67.9077,18.2736,0.1297,0.1286,0.1428,0.1127,9.6101,1
2.9323,66.6893,17.0061,0.0636,0.1092,0.057,0.1728,9.0358,1
2.6235,59.5185,10.5185,0.0412,0.0576,0.1975,0.0206,3.2634,1
3.1901,68.5179,18.7202,0.1234,0.1588,0.1204,0.1327,9.8588,1
2.9765,64.3897,15.2629,0.1004,0.1176,0.1136,0.1519,7.1202,1
3.0524,67.2671,18.2612,0.1411,0.1518,0.1408,0.1768,9.7997,1
2.9472,64.5596,15.5596,0.1251,0.1564,0.0827,0.0814,7.5277,1
```

Şekil 3. 6. Karıştırılmamış APK Özellikleri (SE)

Yapılan dönüşümlerin ardından weka yazılımı üzerinden çeşitli makina öğrenimi algoritmaları üzerinden hesaplamalar yaptırılmıştır. Birtakım algoritmalar denenmiş olup en yüksek doğruluk oranı ise ağaç temelli algoritmalar ile rastgele temeline çalışma prensibine sahip algoritmalarda elde edilmiştir. Kullanılan algoritmalar çizelge 3.3'te gösterilmektedir. Kod karıştırıcısı tespit edici için her iki karıştırma tekniği için de 19 adet algoritma üzerinde denemeler yapılmıştır. En çok başarı oranları ise ağaç ve rastgele bazlı çalışan algoritmalar da görülmüştür. Bunun sebebi ise dizi şifreleme karıştırma tekniği için APK dosyalarından çıkarılan özelliklerde ağaç temelli bir yol izlenmiş olmasıdır.

Çizelge 3. 3. Weka Yazılımı Üzerinden Kullanılan Makine Öğrenimi Algoritmaları.

Tanımlayıcı Yeniden Adlandırma	Dizi Şifreleme
REPTree	REPTree
Random Tree	Random Tree
Random Forest	Random Forest
M5P Tree	M5P
M5 Rules	M5 Rules
Decision Table	Decision Table
Regression by Discretization	Regression By Discretization
Random SubSpace	Random SubSpace
Randomizable Filtered Classifier	Randomizable filtered Classifier
Random Comittee	Random Comittiee
Bagging	Bagging
Additive Regession	Additive Regression
Lazy LWL	KStar
Lazy KStar	Lazy IBK
Lay IBK	SMOreg
SMOreg	Simple Linear Regression
Simple Linear Regression	Multiplayer Perception
.func.Multilayer Perception	Linear Regression

Linear Regression	Gaussian Process
-------------------	------------------

Weka veri madenciliği görevleri için makine öğrenme algoritmalarının bulunduğu bir yapıdadır. Veri hazırlama, sınıflandırma, regresyon, kümeleme ve görselleştirme için araçlar içerir. En önemli kısmı ise açık kaynak kodlu olmasıdır. Bunun yanında weka derin öğrenmeyi de desteklemektedir. (<https://www.cs.waikato.ac.nz/ml/weka/>)

4.BULGULAR

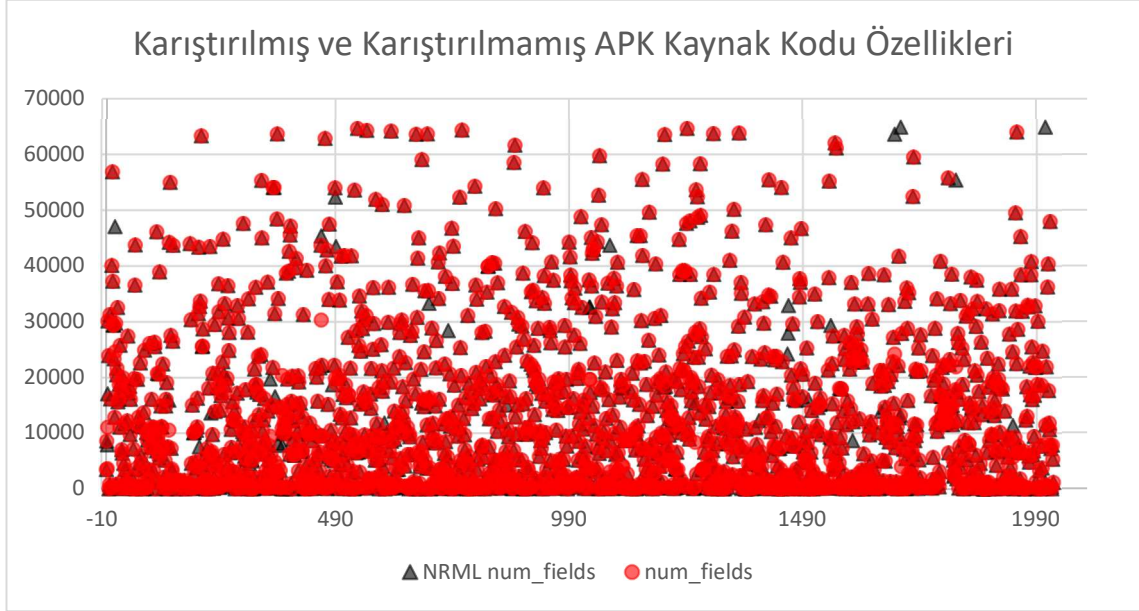
Gizleme/karıştırma yapılmış ve yapılmamış APK dosyalarının tersine mühendislik teknikleri ile elde edilen kaynak kodlarına uyguladığımız yöntemler sonucunda özellikler elde ettik. Bu özellikler makine öğrenimi yöntemlerine uygulanarak sınıflandırma çalışması gerçekleştirilmiştir.

Sistemin eğitilmesi ve test çalışmalarında kullanılmak üzere f-droid sitesinden 2800 adet APK indirilmiştir. APK dosyalarında yapılan incelemeler neticesinde karıştırma uygulanmamış olan 2192 APK tarafımızdan karıştırma ve test kullanılmak üzere tespit edilmiştir. Allatori aracı ile temiz olarak işaretlediğimiz APK'lara tanımlayıcı yeniden anlamlandırma ve metin şifreleme yöntemleri uygulamıştır. Aşağıdaki şekillerde temiz ve Allatori aracı ile karıştırılmış APK'lara ait çıkarılmış olan öz niteliklere yer verilmiştir.

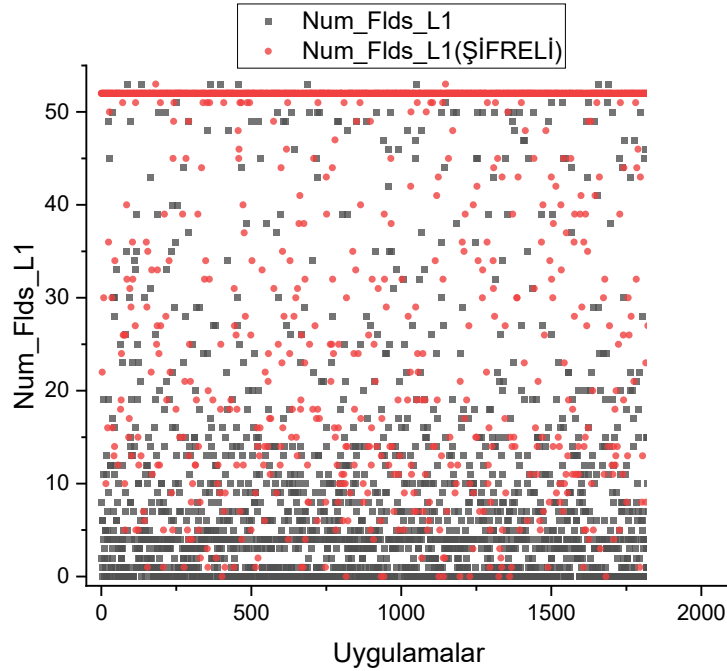
Değişken isimleri normal uygulamalarda “kenarSayisi” ve “counter” sayısı gibi genellikle üçten fazla sayıda harften meydana gelmektedir. Karıştırma uygulandığında ise a, a1 ve ab1 gibi değişken isimlerine dönüştürüldüğü gözlemlenmiştir. Geliştiricilerin farklı desenlerde değişken isimleri üretebilme imkanları her zaman vardır. Biz çalışmamızda varsayılan olarak kullanılan karıştırma yöntemlerini baz aldık. Dolayısı ile değişken isimlerinin kaç karakterden meydana geldiği ayırt edici bir özellik olarak karşımıza çıkmaktadır. Şekil 4.1’de APK’ların toplan alan adı sayıları grafikte gösterilmiştir. Görüldüğü üzere çoğu noktacık üst üste gelmiş, bu durum karıştırma işleminin esasında içerik olarak değişikliğe yol açmadığına dair en önemli göstergelerden birisidir. Bu durumun farkedilebilmesi açısından grafik kasıtlı olarak ayrılmamış olup noktacıkların üst üste geldiğinin görülmesi istenilmektedir.

Şekil 4.2’de ise APK dosyalarının kaynak kodlarında bulunan tek karaktere sahip alan adlarının sayısını gösteren grafik bulunmaktadır. Görüldüğü üzere 52 sayısının üzerine çıkamamıştır bu durum Androdet mekanizmasının sayaçlarında bulunan bir kısıtlama ile oluşmuş bir durumdur. İngiliz alfabesinde bulunan 26 karakter ve büyük harfler ile 52

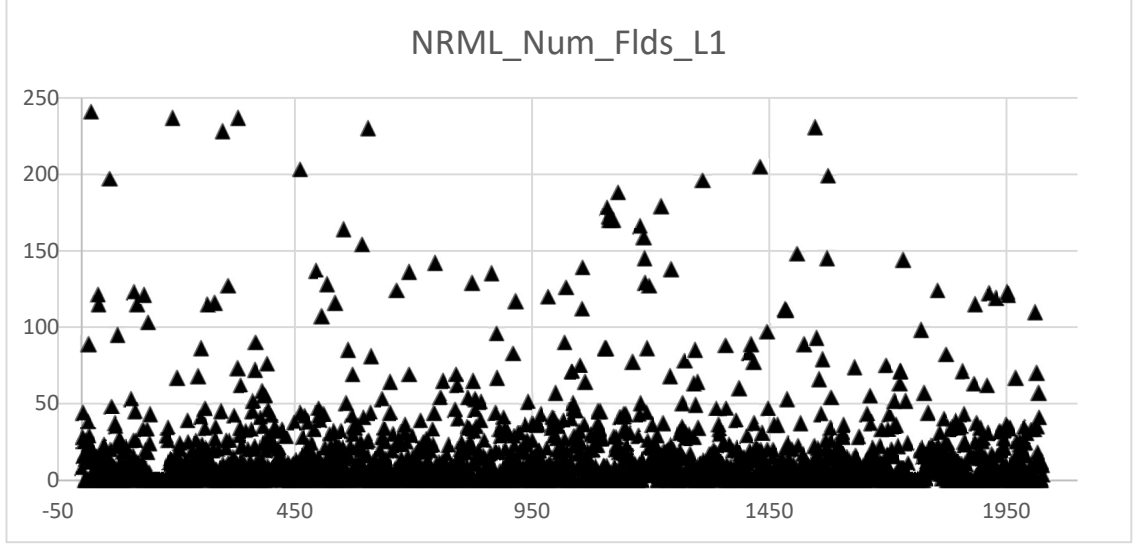
sayısında sabit kalacak şekilde düzenlenmiş olduğundan 52 adet özgün alan adı tespiti yapılarak gösterilmiş durumdadır.



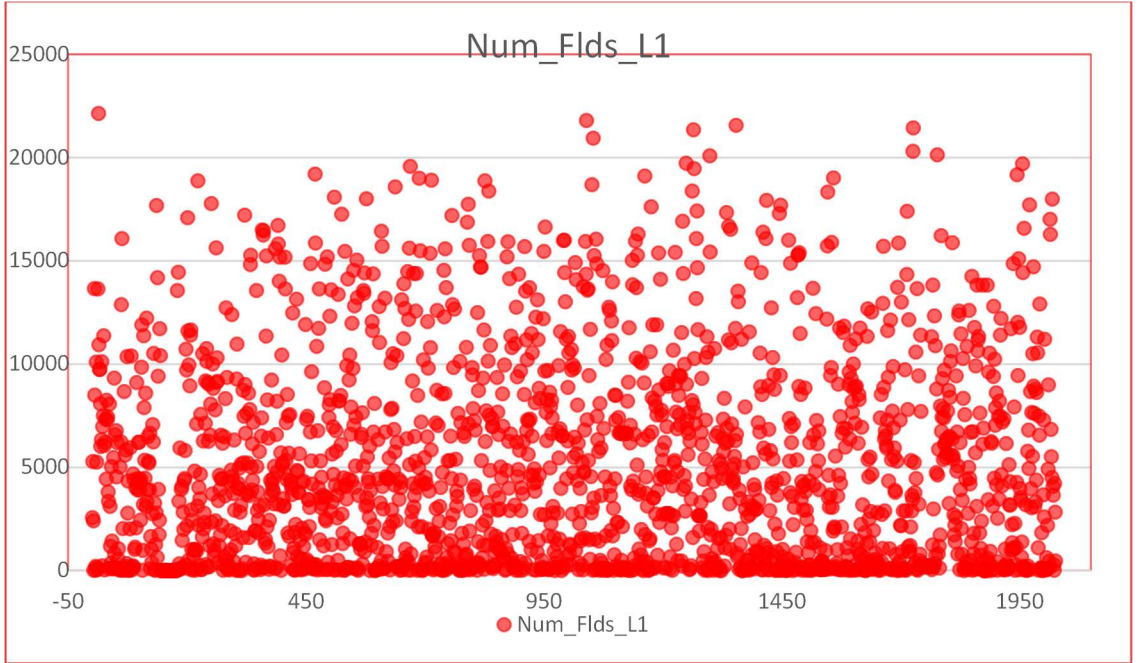
Şekil 4. 1 Normal ve Karıştırılmış APK Kaynak Kodu Toplam Alan Sayıları



Şekil 4. 2. Normal ve Karışık APK'ların Tek Karakterli Alan Adı Sayıları

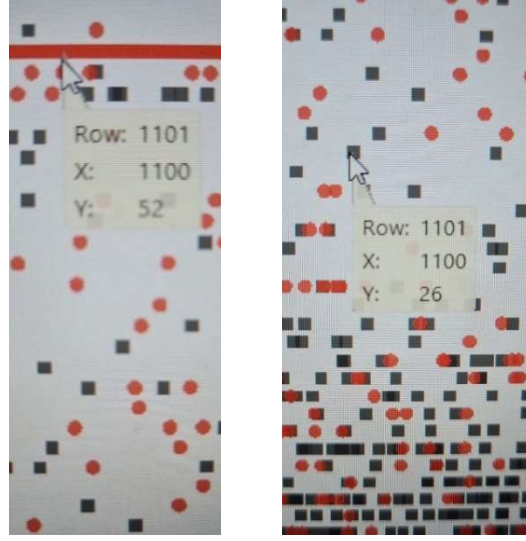


Şekil 4. 3. Normal APK'ların Tek Karakterli Alan Adları Grafiği



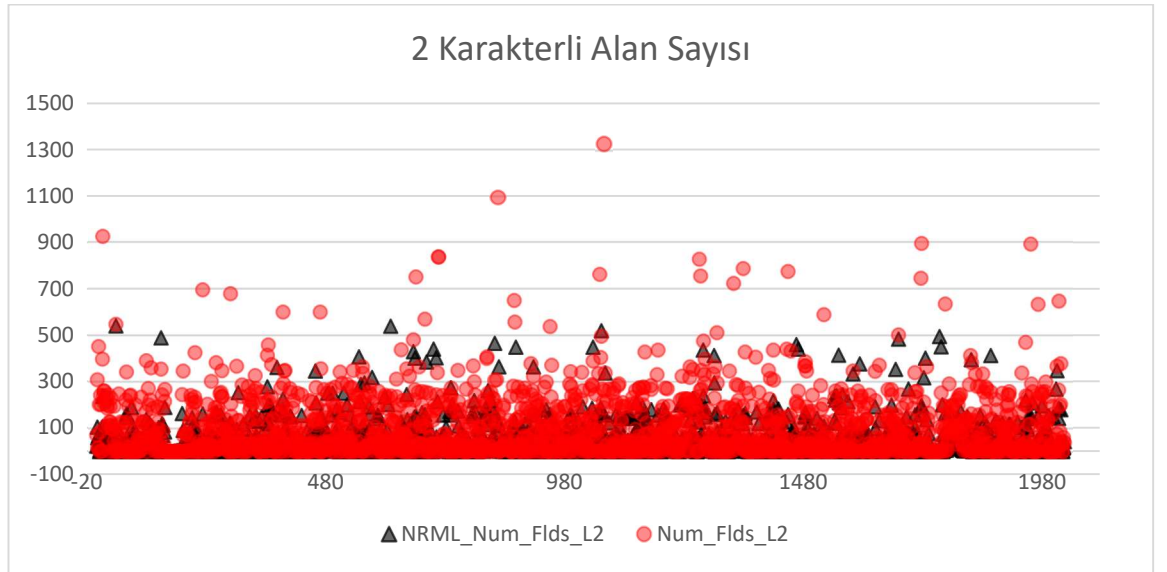
Şekil 4. 4 Karışık APK'ların Tek Karakterli Alan Adları Grafiği

Şekil 4.3'te görüldüğü gibi üçgenlerle gösterilen veriler altında ki Şekil 4.4'te ki kırmızı noktacıklarla gösterilen verilerden bir hayli düşük değerler almıştır. Bunun sebebi APK dosyalarında karıştırma işlemi yapıldığında birden fazla karakterle isimlendirilen alanların karıştırma işlemi neticesinde tek karakterle isimlendirilmesidir. Şekil 4.2'de gösterilenden farkı ise her özgün isme sahip alan adlarını bir kez göstermekten kaçınılarak her bir tek karaktere sahip alan adını sayaçta işleterek elde edilmesidir.



Şekil 4. 5. Aynı APK Dosyasının Şifrelenmiş ve Şifrelenmemiş Halinin Değerleri

Şekil 4.5'te gösterilen ise kısıtlama halinde bulunan aynı APK dosyasının karıştırma yapılmadan önce ki ve sonra ki halinin gösterimidir. Burada gerçek değer çok daha yüksek iken androdet kısıtlaması sebebiyle bu yanlış değerlendirme oluşmuş durumdadır.

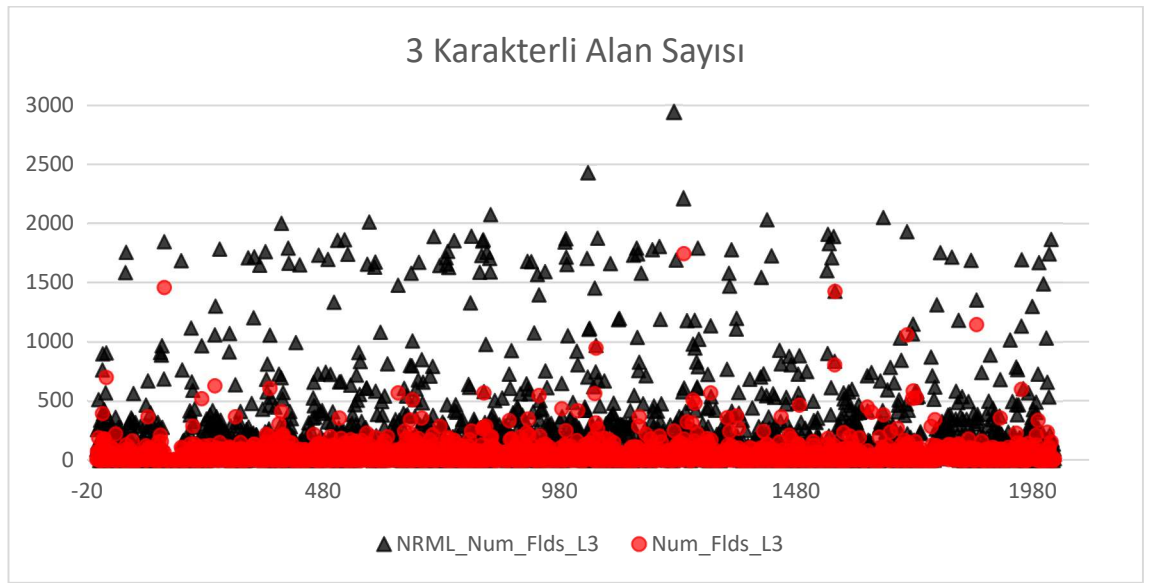


Şekil 4. 6. Normal ve Karışık APK'ların İki Karakterli Alan Adı Sayıları

Şekil 4.6 ise iki karakterle adlandırılan alanların sayısal olarak gösterimini içermektedir. Daha önce ki gösterime göre ise değerlerin biraz daha birbirine yakın olduğu göze çarpmaktadır. Şekil 4.6'da arada ki fark daha da azalmakla birlikte 3 karakterle adlandırılan alanların sayılarında normal durumda bulunan APK'ların bu grafikte birlikte

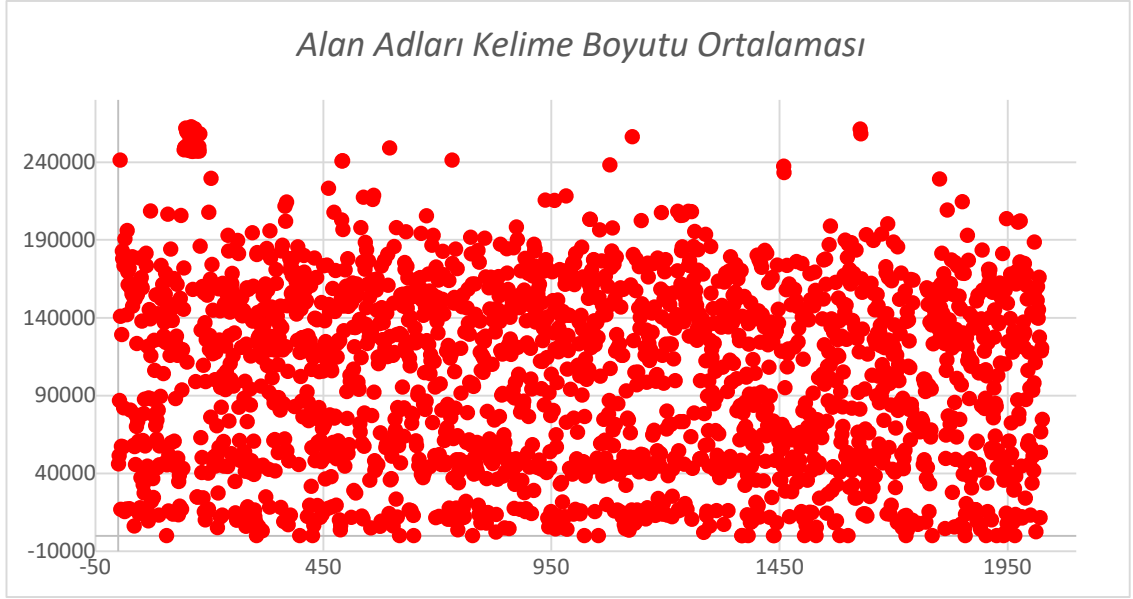
daha fazla sayıda alan adına sahip olduğu da gözükmemektedir. Grafik, farkın anlaşılabilmesi adına ayrılmamıştır.

Grafiklerimizde görüldüğü üzere ise şifreli uygulamaların değerlerinde dalgalı bir grafik olduğu göze çarpmaktadır. Bu duruma sebep olan ise karıştırma yapılırken bir noktadan sonra alfabede bulunan harf sayısı yeterli gelmeyip yanına rakamların eklenmeye başlanmasıdır. Bu durum a0, a1, b0, b1 vb. şeklinde devam eden karıştırılmış tanımlayıcılardır.



Şekil 4. 7. Üç Karaktere Sahip Alan Adı Sayısı Grafiği

Şekil 4.7’de görüldüğü üzere 3 karaktere sahip alan adlarında karıştırılmamış durumda olan APK’ların kaynak kodlarında karıştırılmış durumda olanlara göre daha fazla sayıda alan adı olduğu göze çarpmaktadır. Şekil 4.3 ve şekil 4.4’te grafikte ki verilerden farklı durumda olduğu ve orada ki karıştırılmış durumda olan APK’ların daha fazla sayıda tek karakterli alan adına sahipken burada ki durumda normal durumda olan APK’ların sayısı daha fazla durumdadır. Bu durumun altında ise doğal durumda da 3 karaktere sahip alan adlarının bulunabileceği gerçeği yatmaktadır. Bu bağlamda değerlendirme açısından diğer veriler kadar efektif sonuçlar veremeyebilir.

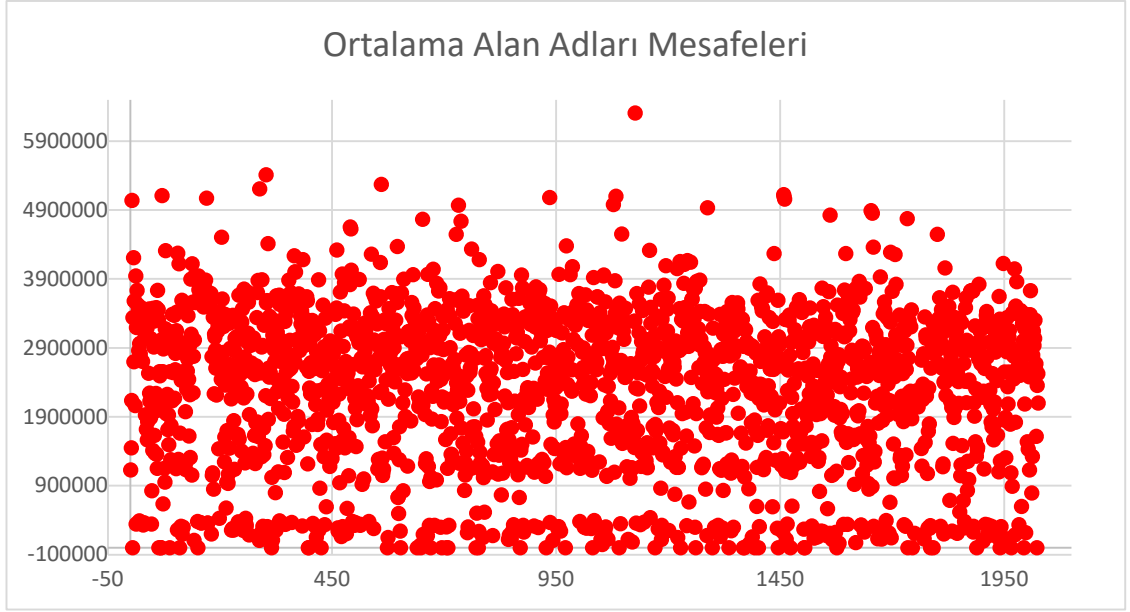


Şekil 4. 8. Alan Adları Kelime Boyutu Ortalaması (karıştırılmış)



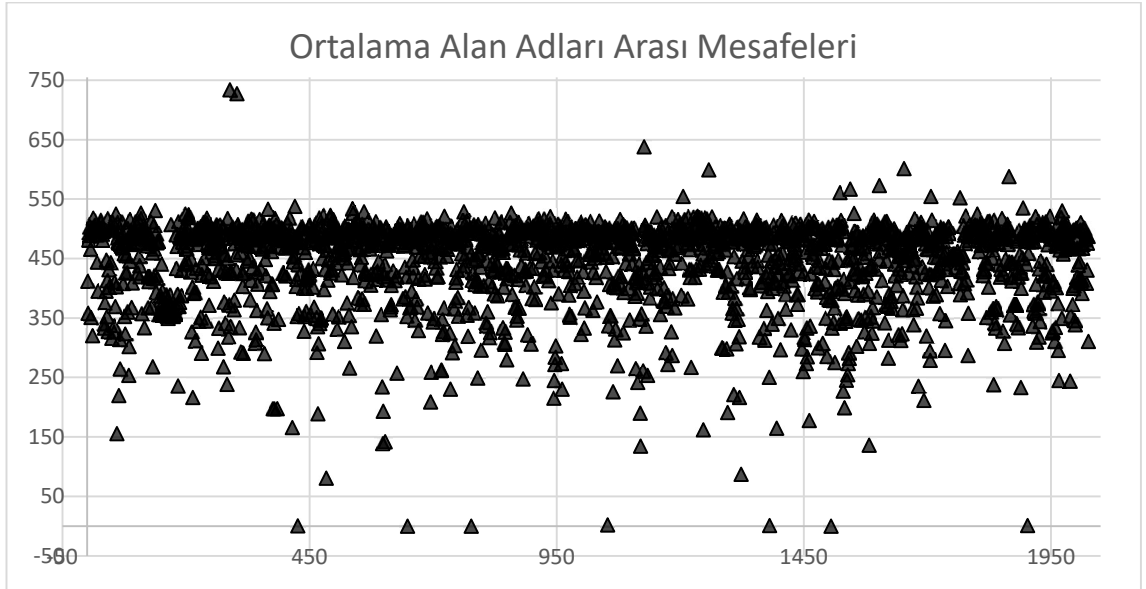
Şekil 4. 9 Alan Adları Kelime Boyutu Ortalaması (Normal)

Şekil 4.8’de karıştırılmış durumda bulunan APK dosyalarının ortalama alan adları kelimelerinin uzunluk olarak ortalaması gösterilmektedir. Karıştırılmış durumda ki APK dosyalarının ortalaması oldukça geniş bir yayılım alanı bulmuştur, bir sonraki grafik olan Şekil 4.9’de ise normal durumdaki APK’ların grafiği bulunmaktadır. İki grafiğin değerleri ve yayılım alanları birbirinden oldukça farklıdır. Bu bağlamda verileri ayrı ayrı grafikletirmek farkın görülebilmesi açısından daha faydalı olacağından ayrılmıştır.

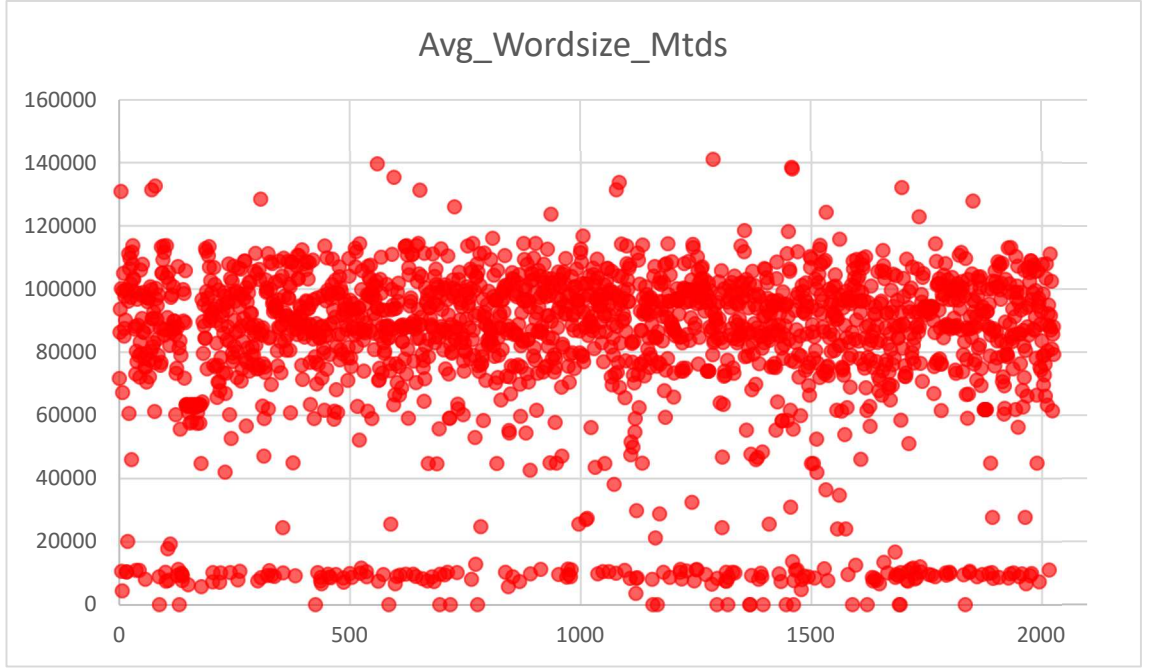


Şekil 4. 10 Alan Adları Mesafe Ortalaması (Karıştırılmış)

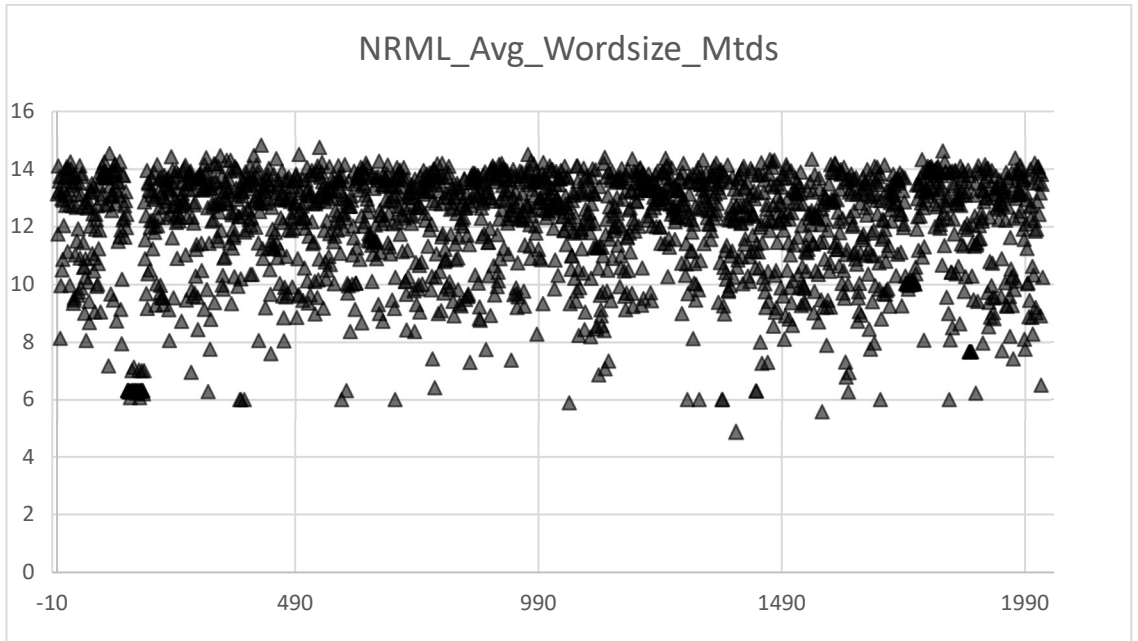
Şekil 10 ile Şekil 11 grafiklerinin ayrılması daha uygundur, veriler arasında ki uçurum çok bariz bir şekilde görülmektedir. Karıştırma işleminin ardından alan adlarının uzunlukları değiştiğinden ötürü ASCII mesafeleri de artmış bulunmaktadır.



Şekil 4. 11. Normal APK'ların Alanlar Arasındaki ort. ASCII mesafesi.

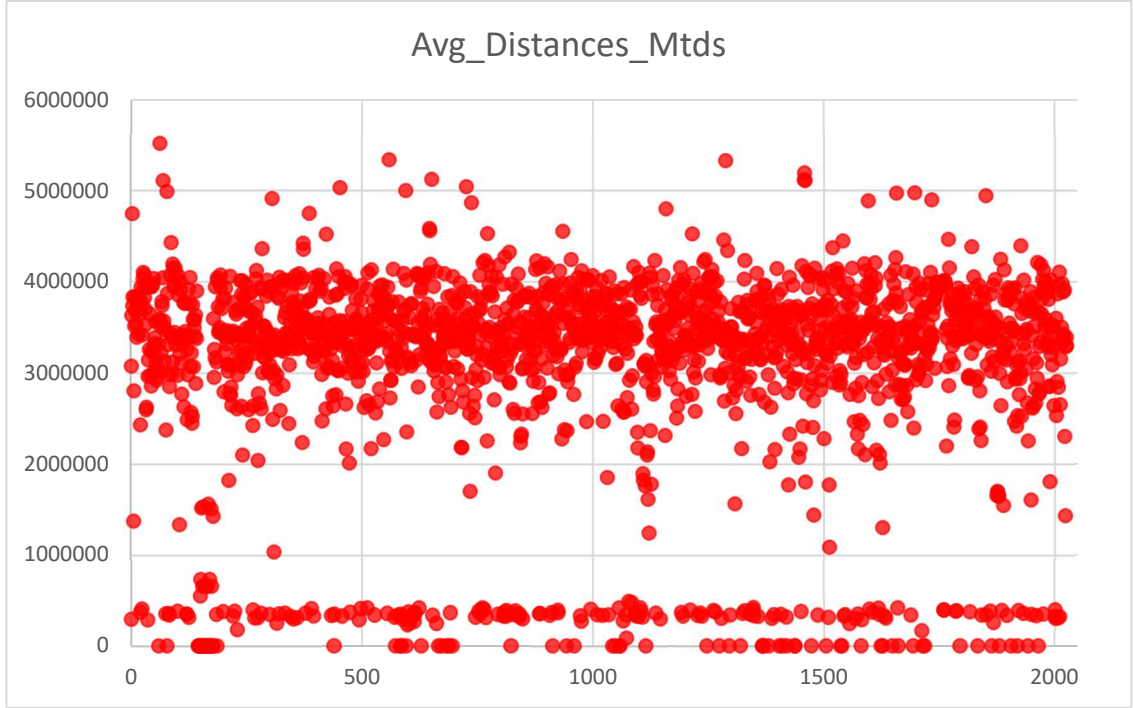


Şekil 4. 12. (Karıştırılmış) Ortalama Metot Kelime Sayısı

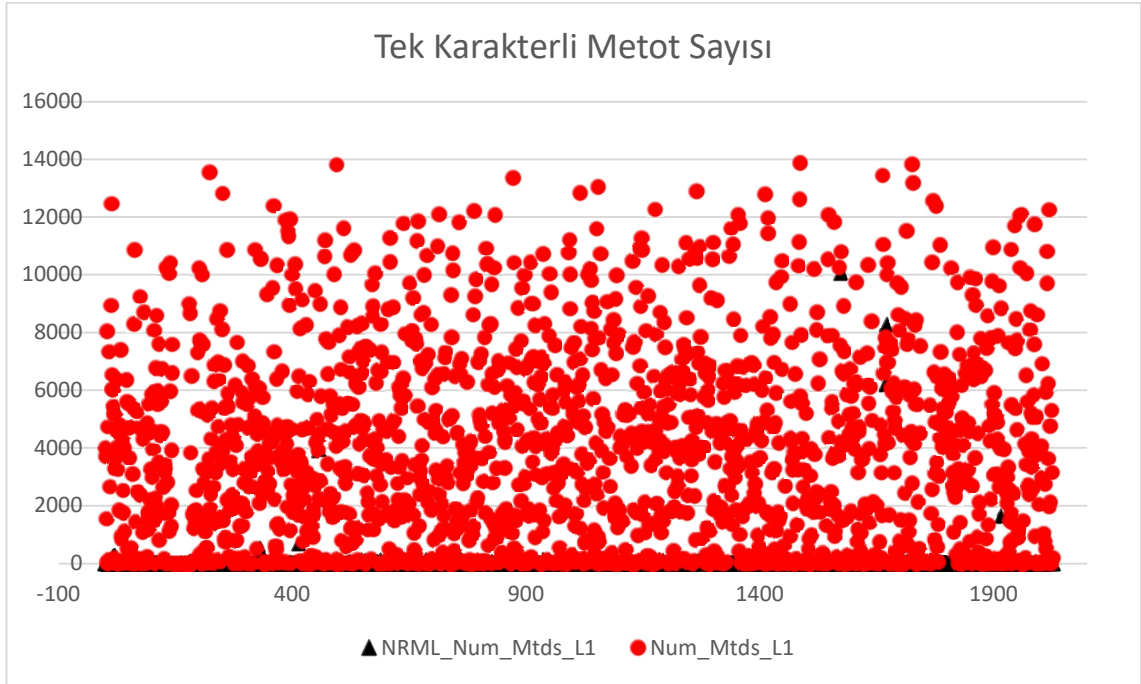


Şekil 4. 13 (Normal) Ortalama Metot Kelime Sayısı

Şekil 4.12 ile Şekil 4.13'te APK başına düşen ortalama metot sayıları görülmektedir. görüldüğü üzere karıştırılmış durumda olanlarda yine sayı aşırı derecede artmış durumdadır. Buna etki eden ise karıştırma işleminin kendisidir. Karıştırma işlemiyle birlikte metot karakter ve sayısında meydana gelen değişimden ötürü bu fark ortaya çıkmaktadır.



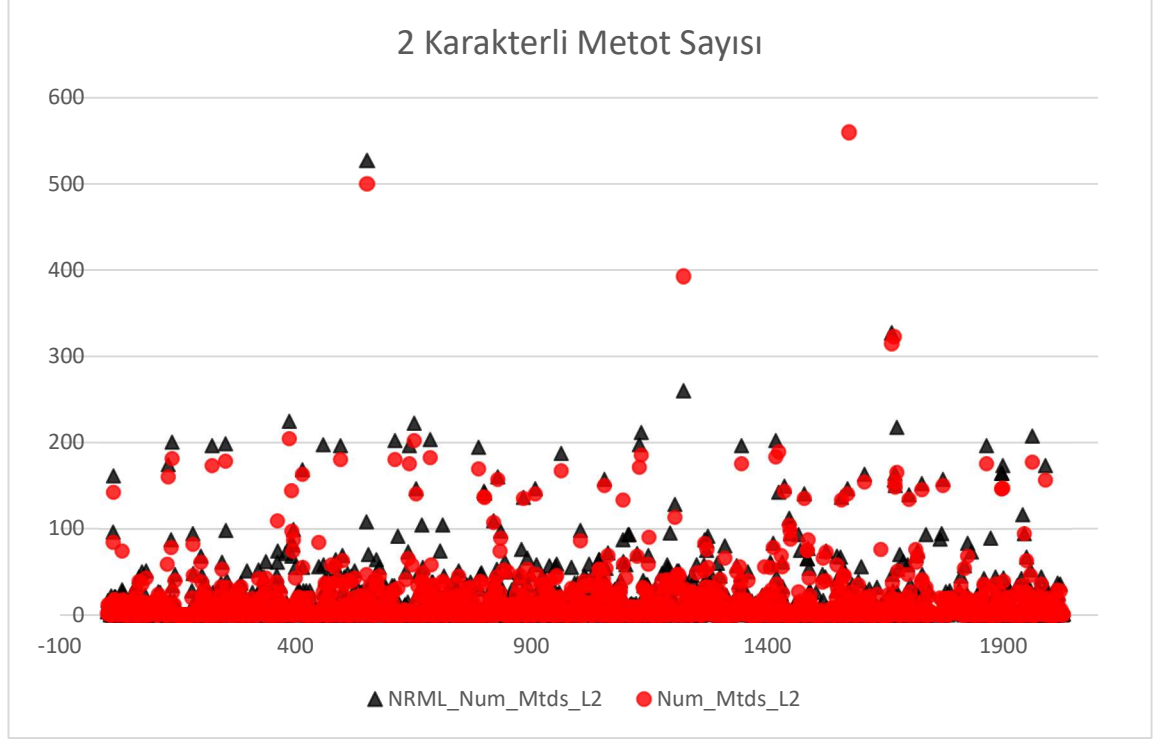
Şekil 4. 14. (Karıştırılmış) Ortalama Metotlar Arası Mesafe



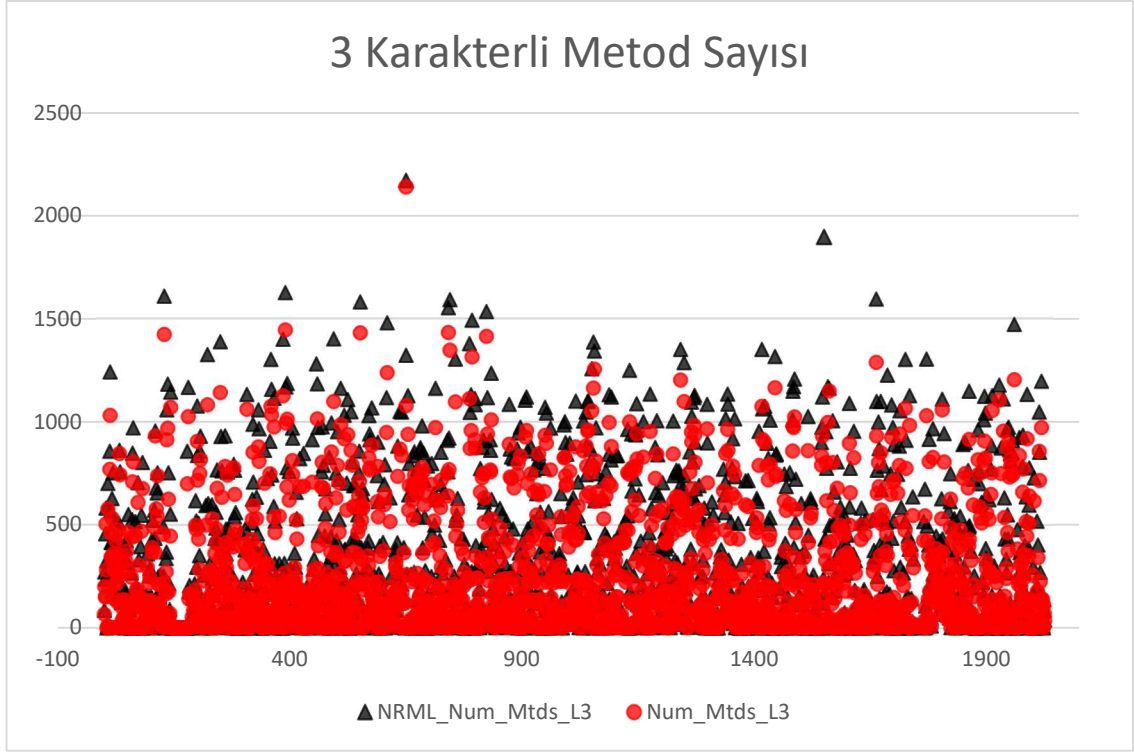
Şekil 4.15. Tek Karakterli Metot Sayısı

Şekil 4.15'te görüldüğü gibi tek karakterli metot sayılarında daha önce görülen durum yine gözükmemekte olup karıştırılmış durumda veriler bir hayli artmaktadır.

Öte yandan Şekil 4.16’da görüldüğü üzere yine daha önce karşılaşılan durum yaşanmış olup 2 veya 3 karakterli metot sayılarında arada ki fark oldukça azalmış durumdadır. Bu bağlamda bazı karıştırılmamış APK’larda metot sayılarının karıştırılmış olana kıyasla daha fazla olduğu grafikte açıkça gözlemlenmektedir.

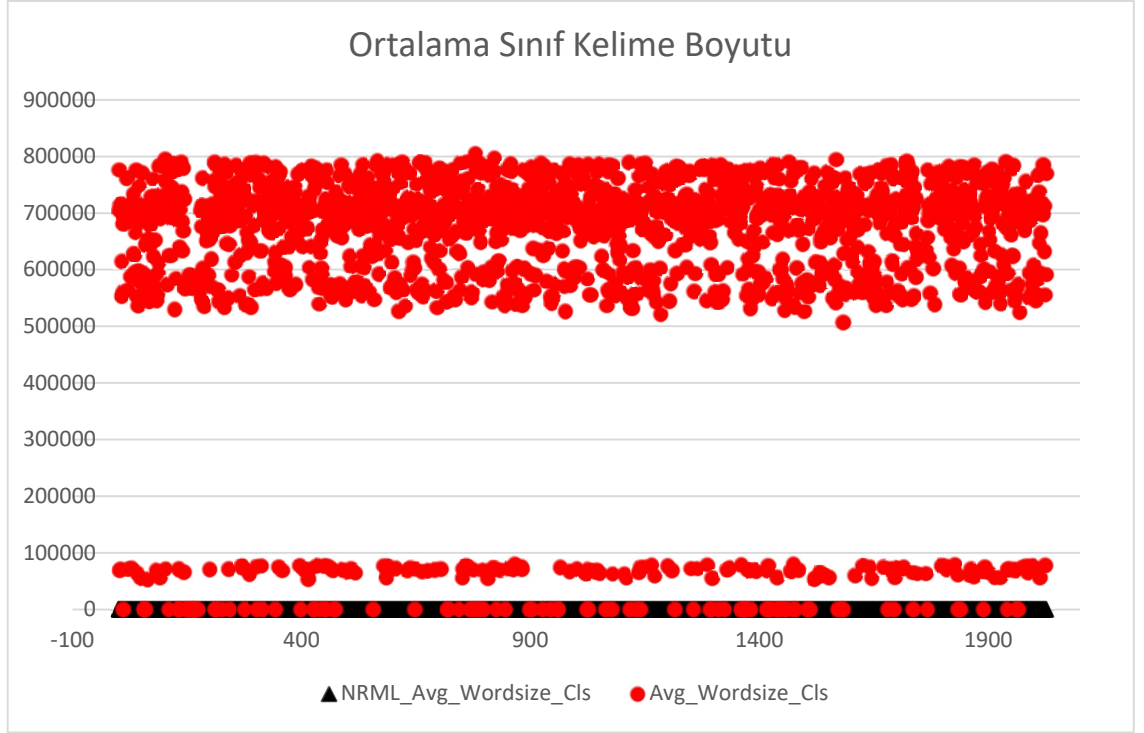


Şekil 4. 16. Çift Karakterli Metot Sayısı Grafiği

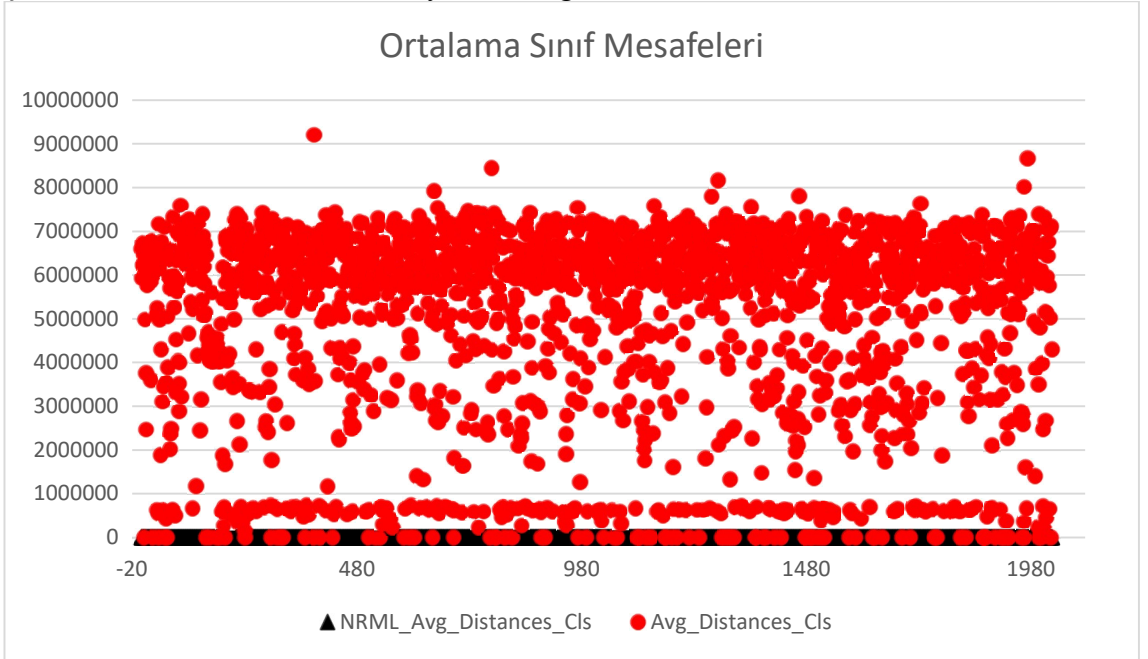


Şekil 4. 17. Üç Karakterli Metot Sayısı Grafiği

Şekil 4.17’de görüldüğü üzere daha önce bahsedilen durum göze çarpmaktadır. Burada bahsedilen durum, karıştırılmamış APK dosyalarının kaynak kodlarından elde edilen verilen grafikte karıştırılmış olanlara kıyasla daha yüksek sayıda olmasıdır.

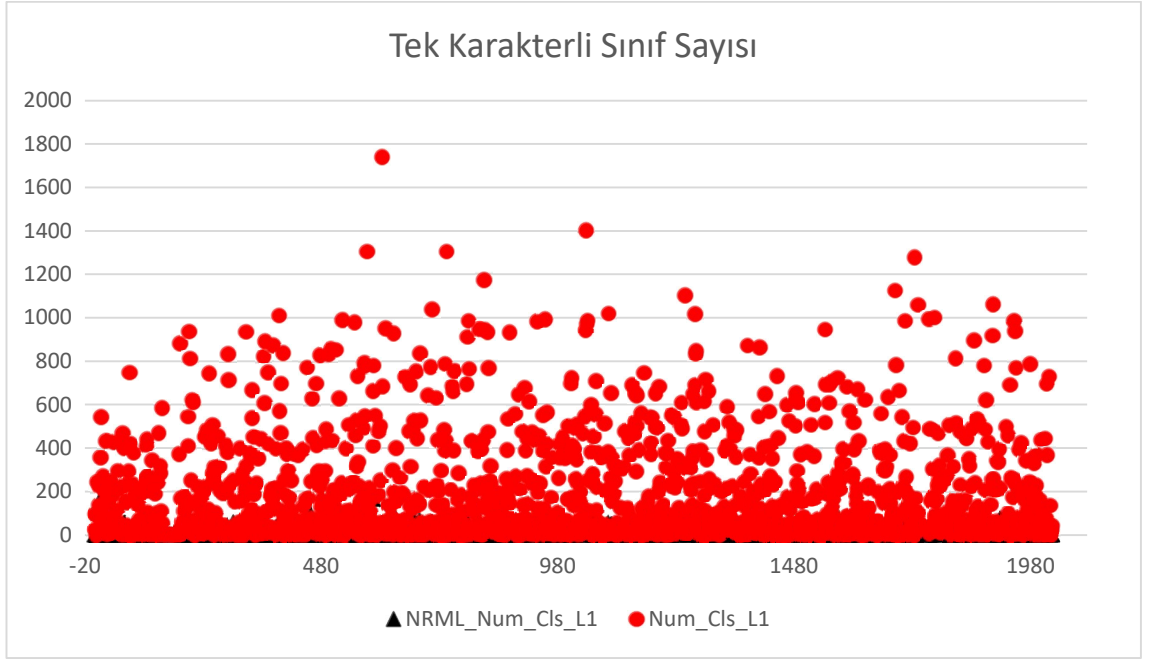


Şekil 4. 18. Ort. Sınıf Kelime Boyutu Grafiği

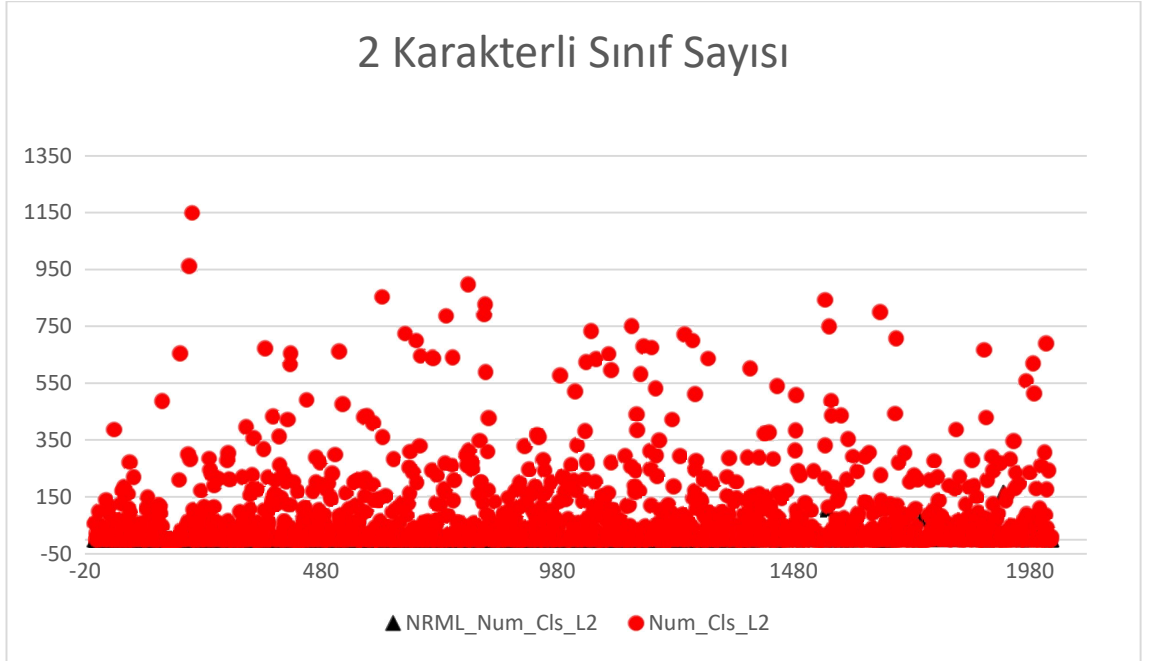


Şekil 4. 19. APK'ların Ardışık Sınıflar Arasındaki Ort. ASCII Mesafesi

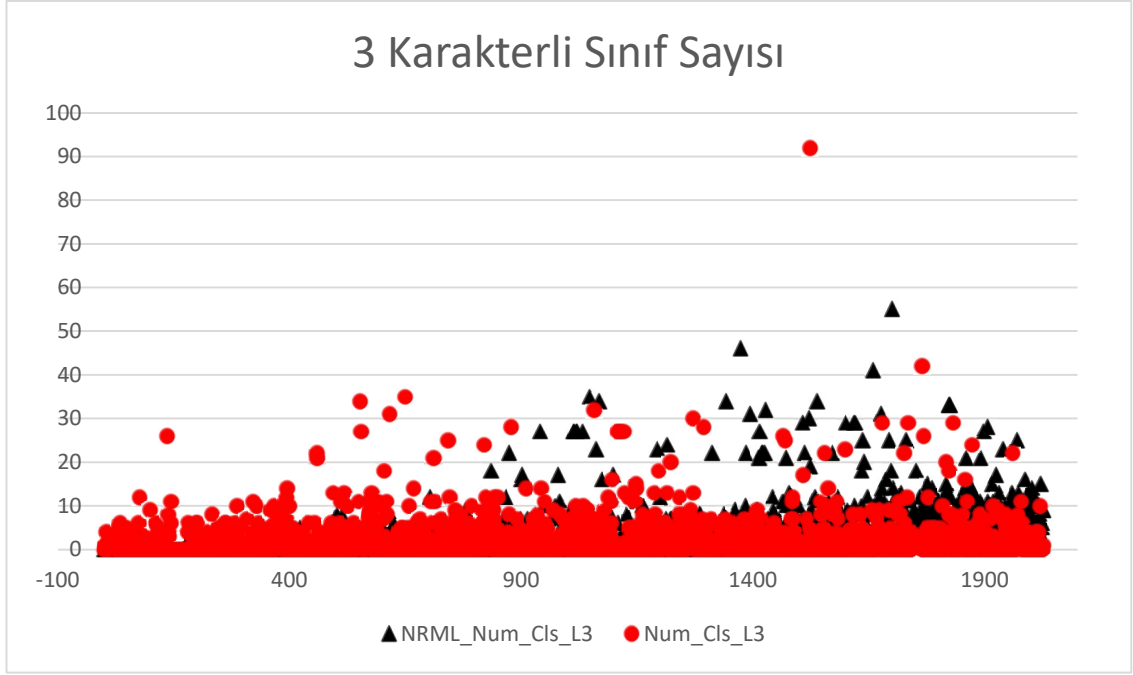
Verilerde ki değişim benzerlik gösterdiğinden ortalamalar grafiği de hemen hemen benzer durumda şekillenmiştir. Bu bağlamda Şekil 4.18 ile Şekil 4.19'da görülmektedir oldukça da benzer grafik içeriğine sahiptirler.



Şekil 4. 20. Normal ve Karışık APK'ların Sınıf Uzunluğunun Özellikleri

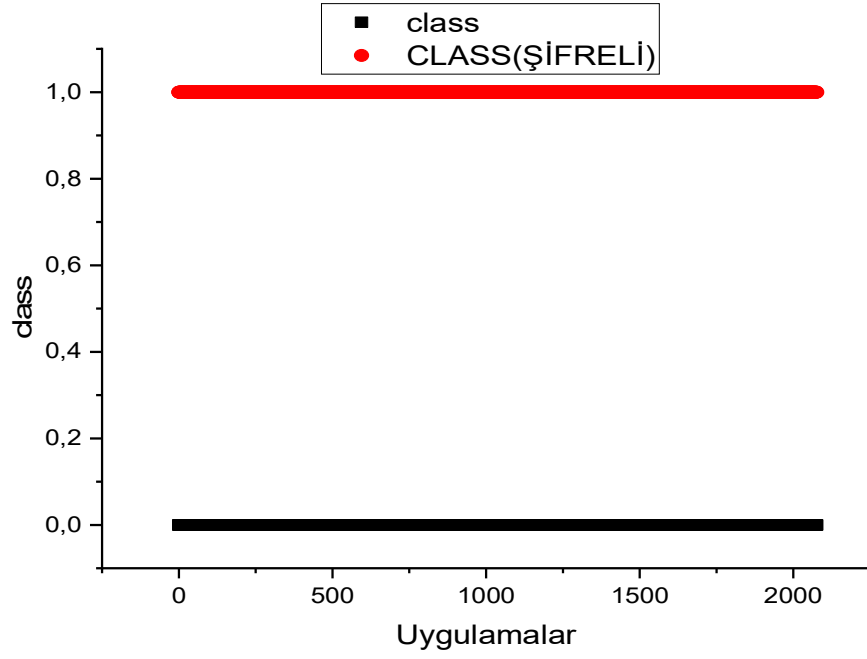


Şekil 4. 21. İki Karakterli Sınıf Sayısı Grafiği



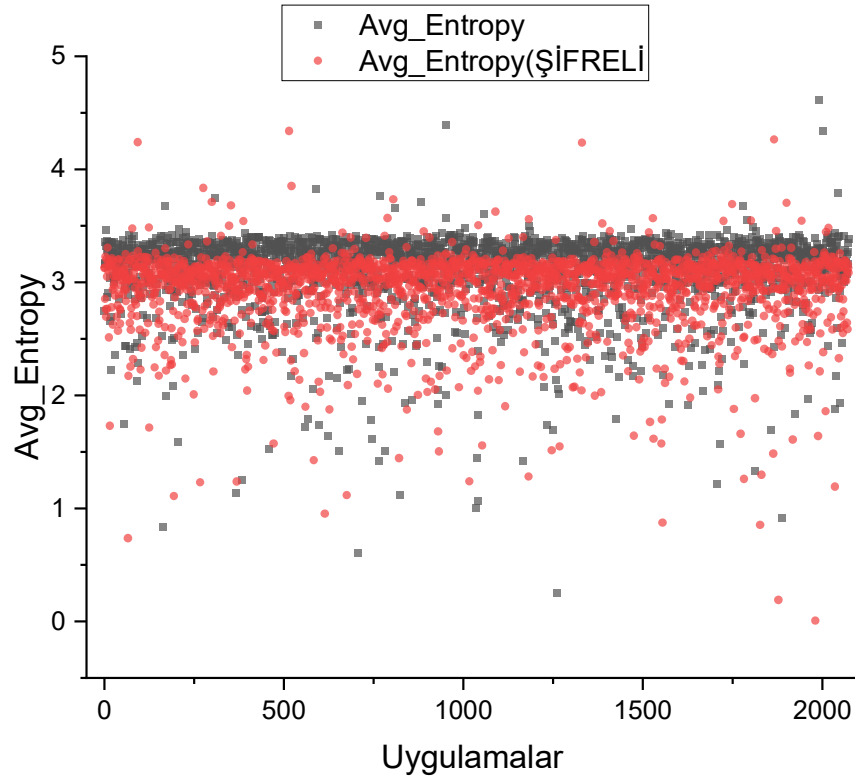
Şekil 4. 22. Üç Karakterli Sınıf Sayısı Grafiği

Benzer durum sınıf sayısı grafiklerinde de göze çarpmaktadır. Tek karakterli olanlarda karıştırılmış olan bir hayli yüksek iken iki karakterli olanlarda fark biraz daha azalmış olup 3 karakterli olanlarda veriler birbirine oldukça yakındır.



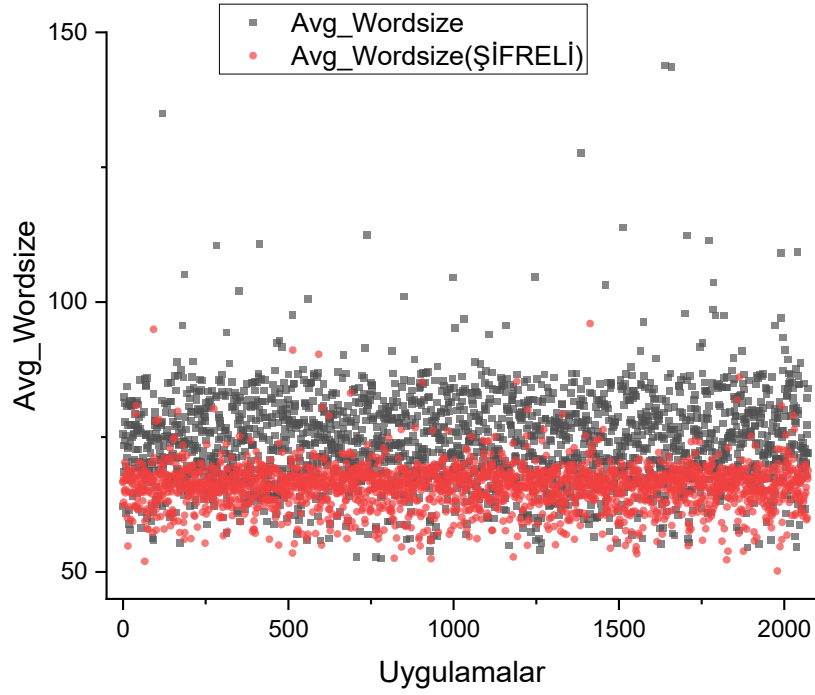
Şekil 4. 23. Sınıf Belirlenmesi(Etiketleme) Grafiği

Grafiklerde tanımlayıcı yeniden adlandırma gizleme tekniği hakkında çıkarılan özelliklerin karşılaştırılmaları mevcuttur. Anlamli olması bakımından ve kümeleşmelerin daha net bir şekilde görülebilmesi amacıyla tüm grafiklerde noktacıklar kullanılmış olup, karıştırılmış ve karıştırılmamış durumda olan veriler için üçgen ve daire şekilleri ile fark ortaya konulmaya çalışılmıştır.. Şekil 4.23'e kadar olanlar tanımlayıcı yeniden adlandırmaya ait olan özellikler iken Şekil 4.24 itibari ile olan grafikler dizi şifreleme ile ilgilidir. Tanımlayıcı yeniden adlandırma tekniği ile gizlenen uygulamalarda ölçülebilen mesafeler genel olarak şifresiz hallerinden daha fazla olduğu görülmektedir.



Şekil 4. 24. APK'ların Dizilerinin Ortalama Karmaşıklık Değerleri

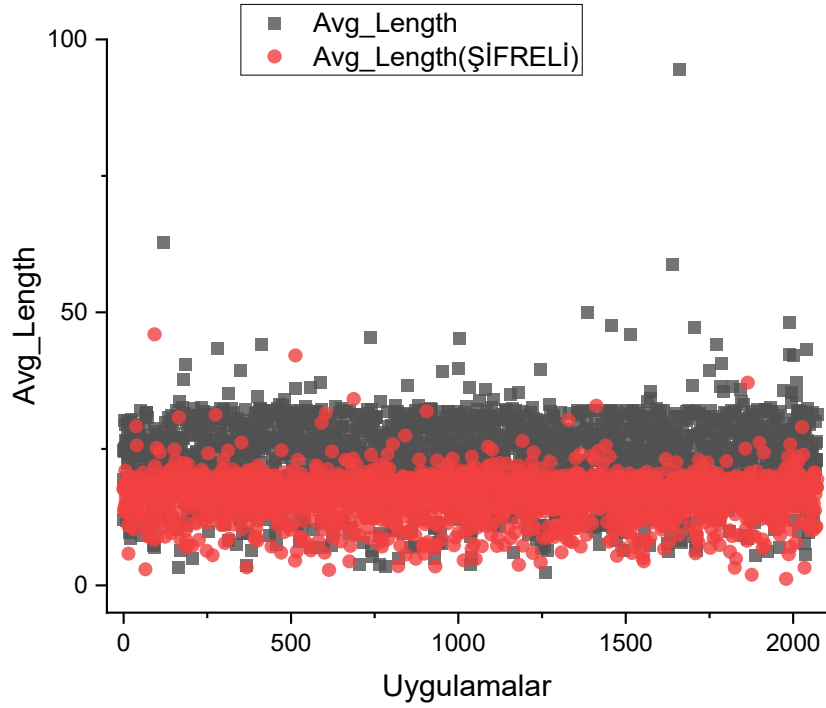
Şekil 4.24'de görülen grafikte dizilerin ortalama karmaşıklık değerleri görülmektedir. Grafiklerde gösterilmeye çalışılan özellikler elde edilirken; ortalama karmaşıklık, ortalama sözcük boyutu, ortalama dize uzunluğu, ortalama eşittir (=) sayısı, tireler (-), eğik çizgiler (/), ve artılar (+) ve son olarak da toplam tekrarlayan karakterlerden oluşur. Bu özellikleri ise önemlerine göre sıralayan ve puanlayan bir özellik seçilimi algoritmalarından olan ağaç tabanlı algoritmalarından faydalanılmıştır. (Bui et al., 2019)



Şekil 4. 25. Normal ve Karışık APK’ların Dizilerinin Ort. Kelime Boyutları

Şekil 4.25’da ise dizilerin ortalama kelime boyutlarının verilerinin grafiğe aktarılmış değerleri görüntülenmektedir. Görüldüğü üzere şifrelenen APK dosyalarının kaynak kodlarından elde edilen verilerin daha düşük değerlere sahip olduğu anlaşılmaktadır. Bu veriler ışığında şifrelenen kodların daha düşük uzunlukta olduğu anlaşılmaktadır. Bunun sebebi olarak ise karıştırma işlemleri esnasında tanımlayıcı adlarının orijinal halinden daha kısa olan anlamsız harf ve rakam kombinasyonu ile değiştirilmesi söylenebilir.

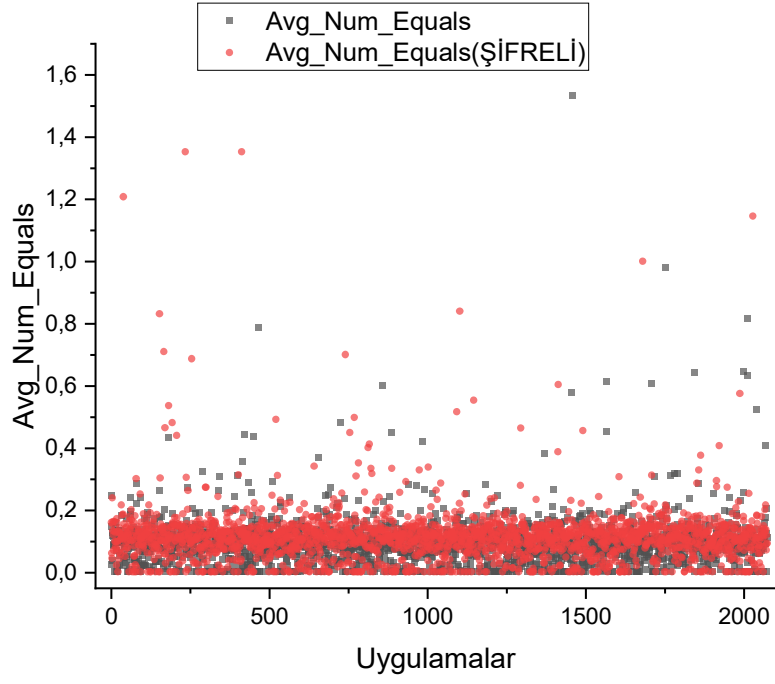
Örneğin bir değişkenin adı “counter_people” iken karıştırma işlemi neticesinde yeni adı sadece “a” olmuş olabilir. Bu kısaltmaların ardından ise ortalama kelime boyutu veya uzunluğu da azalmıştır.



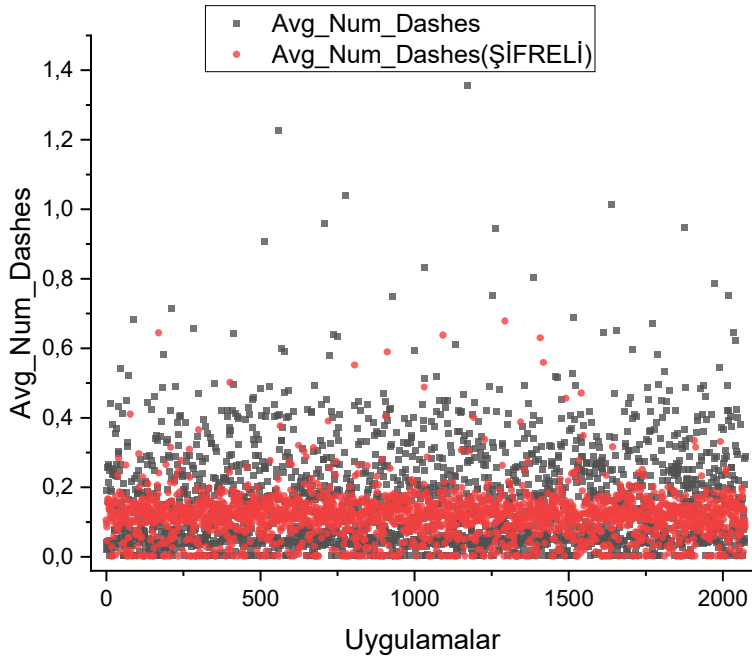
Şekil 4. 26. Normal ve Karışık APK'ların Ortalama Uzunlukları

Şekil 4.26'de görüldüğü üzere şifrelenmiş (karıştırılmış) APK dosyalarının kaynak kodlarının ortalama uzunluğu genel olarak 15-25 bandında olduğu olarak göze çarparken karıştırılmamış APK dosyalarının kaynak kodu ortalama uzunluğu genel olarak 20-35 bandında olduğu aralığında ortaya çıkmaktadır.

Şekil 4.27'de görüldüğü üzere ise grafikte değerler arasında farklılık olarak göze çarpan bir kümelenme ve yoğunlaşma olmadığı görülmektedir. Bu durumda eşittir değerleri bakımından herhangi bir ayrışım yapmak şu durumda zordur. Belirgin bir farklılık olmadığından analiz işlemlerinde işe yarar bir farklılık yaratmayacaktır.



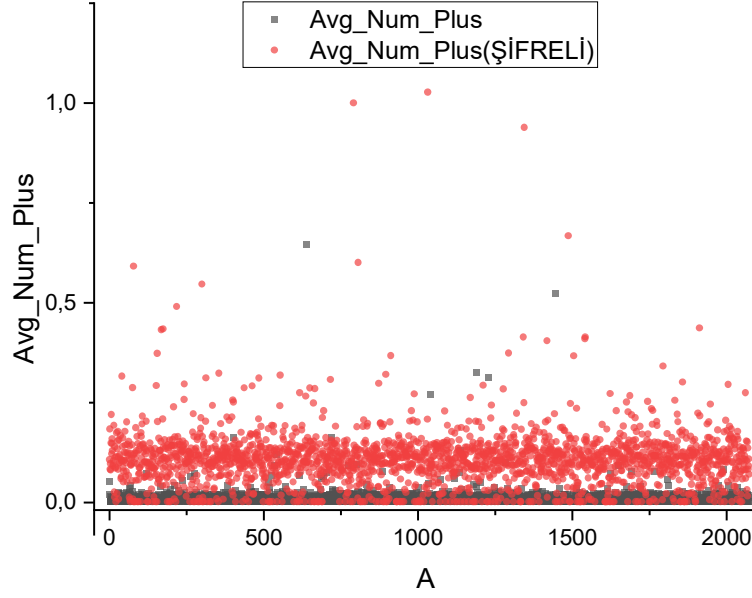
Şekil 4. 27. Normal ve Karışık APK'ların Eşittir Simgesi Ortalamaları



Şekil 4. 28. Normal ve Karışık APK'ların Ortalama Tire (-) Değerleri

Şekil 4.28'de ise tire (-) işaretinin APK dosyalarının kaynak kodlarında ki ortalama değerleri görülmektedir. Şifresiz (karıştırılmamış) APK dosyalarının değerleri daha

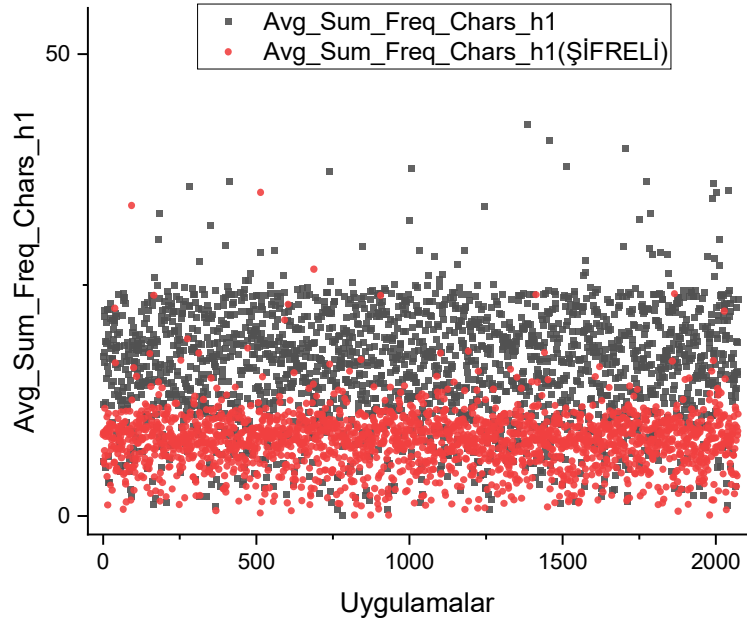
yayılmış halde bulunurken, şifreli (karıştırılmış) APK dosyalarının kaynak kodlarında ki ortalama (-) sayıları daha dar bir alanda sıkışmış olarak görülmektedir.



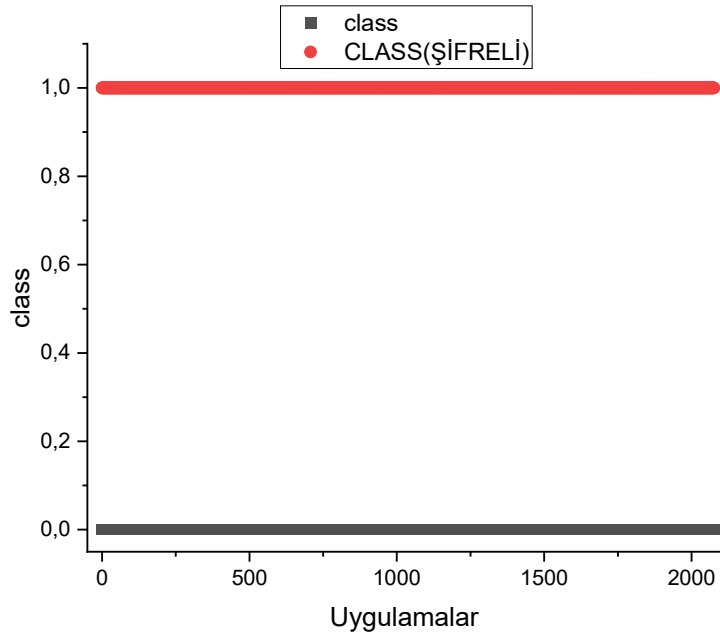
Şekil 4. 29. Normal ve Karışık APK'ların Ortalama Artı (+) Sayısı

Şekil 4.29'te görüldüğü gibi şifreli APK dosyasının kaynak kodlarından bulunan ortalama artı sayıları daha geniş bir aralıkta var olurken, şifresiz APK dosyalarının ise daha dar bir alanda ve tekdüze şeklinde grafikte kendine yer bulmuş vaziyettedir. Bu durum analiz açısından bir nebze elimizi kuvvetlendirmektedir.

Şekil 4.30'da ise tekrar eden karakter dağılımı gösterilmiştir. Yine grafikten anlaşılaacağı üzere şifreli ve şifresiz uygulamalar arasında ki fark meydana çıkmaktadır.



Şekil 4. 30. Normal ve Karışık APK'ların Tekrar Eden Karakterlerin Dağılımı



Şekil 4. 31. Normal ve Karışık APK'ların Class Parametresi Dağılımı

Şekil 4.31'te ise tarafımızca eklenen ve sonucu etkilemesi umulan bir diğer özellik parametresi görülmektedir. Karıştırılmış ve karıştırılmamış olanların kesin bir şekilde ayrıldığı göze çarpmaktadır. Bu bağlamda karıştırılmış olanlara 1 değeri atanırken

karıştırılmamış olanlara ise 0 değeri atanmaktadır. Aynı etiketleme işlemi tanımlayıcı yeniden adlandırma karıştırma yöntemi için de kullanılmıştır.

APK dosyalarının kaynak kodlarının özelliklerinin çıkarılması ile elde edilen veriler grafiğe dönüştürüldüğünde yukarıda gösterilen şekillerdeki düzenler olduğu kümelenmelerin değiştiği fark edilmektedir. Bu veriler Weka yazılımı yardımıyla çeşitli algoritmalar ile modellendiğinde ise aşağıdaki tabloda gösterilen oranlarda doğruluk elde edildiği görülmüştür.

Çizelge 4. 1. Makine Öğrenimi Algoritmaları ile modelleme sonuçları

Tanımlayıcı Yeniden Adlandırma			Dizi Şifreleme		
Algoritma	Öncesi	Sonrası	Algoritma	Öncesi	Sonrası
Rep-Tree	%88.1	%88.8	REPTree	%92.14	%96.58
RandomTree	%91.83	%92.61	RandomTree	%94.81	%99.0
RandomForest	%95.43	%96.19	RandomForest	%95.17	%99.19
m5p	%96.03	%96.03	M5P	%99.86	%99.9
DesicionStump	%84.52	%85.29	Decisionstump	%84.1	%99.73
Rules.M5rules	%97.45	%97.45	M5Rules	%99.87	%92.71
rules.Desiciontable	%97.0	%98.02	DecisionTable	%87.79	%93.21
regressionbydiscretization	%97.6	%98.48	RegressionbyDiscretization	%89.52	%94.96
random_Subspace	%97.39	%97.39	RandomSubSpace	%92.17	%97.1
Randomizable_filtered_classifier	%98.85	%98.85	RandomizableFilteredClassifier	%94.8	%99.55
Random_Comittee	%95.64	%95.64	RandomCommittee	%94.99	%98.26
AdditiveRegression	%97.7	%97.7	Bagging	%94.4	%95.61
LWL	%89.21	%89.21	AdditiveRegression	%92.14	%99.81
Kstar	%98.94	%98.94	Kstar	%88.73	%94.46
Lazy_IBK	%99.08	%99.08	Smoreg	%83.61	%99.94
			SimpleLineerRegression	%97.08	%99.52
			MultuplayerPerception	%93.32	%99.89
			Linearregression	%97.69	%99.94
Ortalama(%)	94.98	95.31		92.89	97.44

Çizelge 4.2’de verilen algoritmalarından ağaç temelli olanlarda daha yüksek bir doğruluk oranı elde edildiği görülmektedir. Bu durumun özellikler çıkarılırken kullanılan ve önemlerine göre sıralayan ve puanlayan ağaç temelli algoritmalarından dolayı olduğu düşünülmektedir. Öte yandan Şekil 4.1’de androdet tarafından oluşturulan ve 52’de kısıtlanan durumda elde edilen doğruluk oranları gösterilmiştir. Şekil 4.2’de ise

kısıtlamanın kaldırılması ve gerçek deęerlere ulaşılması ile elde edilen sonuçlar listelenmiştir. Burada görüldüğü üzere doğruluk oranlarında artış olmaktadır.

5. TARTIŞMA ve SONUÇ

İncelemekte olduğunuz bu tezde, mobil uygulama pazarında önemli bir yere sahip Android cihazların bilgi güvenliği için hayati derecede öneme sahip olan uygulama güvenliğinin bir parçası olan gizleme teknikleri üzerinde çalışılmıştır. Ortaya konan yöntemler ile birlikte bir çatı mekanizması geliştirilmiştir. APK dosyalarının özelliklerinin çıkarılması hususunda Androdet olarak bilinen yapıdan faydalanılmıştır. Özellik çıkarma konusunda belirli tespit yöntemleri kullanılmış olup yapılan geliştirmeler neticesinde özelliklerin çıkarımında bazı farklılıklara gidilmiştir. Doğruluk tespitinde kullanılan algoritmalarda ise farklı algoritmalarla sonuca gidilmeye çalışılmıştır.

Androdet yapısı tek haneli tanımlayıcı alan isimlerini hesaplarken İngilizce alfabede bulunan 26 farklı karakter ve bunların büyük hallerini de sayarak 52 olarak belirlemekteydi. Bu durum model oluşturulurken kullanılan ve gizlenen uygulamaların en önemli ayıraçlarından mahrum kalmasına sebebiyet vermektedir. Nitekim gizleme yapılan uygulamalarda alan adlarının tek haneli karakterlerle değiştirilmesi oldukça yaygın bir durumdur. Bununla birlikte her APK dosyasında birden fazla sınıf ve hiyerarşik olarak her bir sınıfın altında onlarca alan tanımlanabilmektedir. Her bir sınıf için ayrı ayrı düşünüldüğünde tek karakterle yeniden isimlendirilen alan ve sınıf adları gibi tanımlayıcıların sayısı gizlenmiş uygulamalarda binlerce olabilmektedir. Grafiklerden de anlaşıldığı üzere kesin olarak gizlendiği konusunda net bir bilgi verebilmektedir. 52 olarak sayıldığında ise normal APK dosyaları ile arasında ayrım o kadar net olmadığından hatta bazı durumlarda sonucu etkileyecek derecede farklılık yarattığı gözlemlenmiş ve doğruluk oranlarında düşüşler meydana gelmiştir.

Makine öğrenimi yaklaşımlarının bilinen yönlerinden biri olan veri ile beslenmesi ve veri ile beslendikçe doğruluk oranlarının artmasından dolayı bu tezde yaklaşık 2192 adet özgün olmakla birlikte karıştırılmış halleri de düşünülecek olursa 4384 adet uygulama ile sonuçlar elde edilmiştir. Wang ve roundtev tarafından sunulan bir çalışmada bu sayılar oldukça düşüktür ve 282 adet uygulama ile çalışmalar gerçekleştirilmiştir. Az sayıda ki uygulama üzerinden yapılan çalışmalar neticesinde makine öğrenimi yaklaşımları aşırı öğrenmeye doğru kaymıştır. (Yan Wang & Atanas Rountev, 2017)

Bu farklılıklar neticesinde Androdet tanımlayıcı yeniden adlandırma karıştırma tekniğinde %95,31 doğruluk oranı elde edebilmişken yapılan bu çalışmada bu oran %96,94'e yükselmiştir. Benzer şekilde dizi şifreleme gizleme tekniğinde daha önceden %81,41 doğruluk oranı elde edilebilmiş iken yapılan bu çalışmada %97,44 doğruluk oranı elde edilebilmiştir. Önceki çalışmalardan farklı olarak çok sayıda makine öğrenimi algoritması ile veriler test edilmiş olup optimum sonuçlar elde edilmeye çalışılmıştır.

Oluşturulan bu çatı mekanizma ilerleyen çalışmalarda daha büyük veri kümesi ile beslenerek elde edildiğinde daha da yüksek doğruluk oranları elde edilebilecektir.

Gün geçtikçe artan mobilite güvenliğe duyulan ihtiyacı had safhaya çıkarmış bulunmaktadır. Yapılan bu çalışma ile mobil cihazlarda büyük oranda kullanılan android işletim sisteminde bulunan güvenlik problemlerine karşı önlem alınması daha da kolaylaşacaktır. Bununla birlikte kötücül uygulamaların yanı sıra iyi huylu uygulamaların da fikri mülkiyet hakları ve birtakım bilgilerin ele geçirilmesine karşı konulmasına dair çalışmalarda yol göstereceği düşünülmektedir.

,

KAYNAKLAR

- Aonzo, S., Georgiu, G. C., Verderame, L., & Merlo, A. (2020). Obfuscapk: An open-source black-box obfuscation tool for Android apps. *SoftwareX*, 11. <https://doi.org/10.1016/j.softx.2020.100403>
- Apvrille, A., & Nigam, R. (2014). Covering the global threat landscape OBFUSCATION IN ANDROID MALWARE, AND HOW TO FIGHT BACK.
- Arnatovich, Y. L., Wang, L., Ngo, N. M., & Soh, C. (2018). A comparison of android reverse engineering tools via program behaviors validation based on intermediate languages transformation. *IEEE Access*, 6, 12382–12394. <https://doi.org/10.1109/ACCESS.2018.2808340>
- Bacci, A., Bartoli, A., Martinelli, F., Medvet, E., & Mercaldo, F. (2018). Detection of obfuscation techniques in android applications. *ACM International Conference Proceeding Series*. Association for Computing Machinery. <https://doi.org/10.1145/3230833.3232823>
- Balachandran, V., Sufatrio, Tan, D. J. J., & Thing, V. L. L. (2016). Control flow obfuscation for Android applications. *Computers and Security*, 61, 72–93. <https://doi.org/10.1016/j.cose.2016.05.003>
- Biabani, R., Halaghi, M. M., & Ghorbani, Kh. (2016). M5 Model Tree to Predict Temporal Evolution of Clear-Water Abutment Scour. *Open Journal of Geology*, 06(09), 1045–1054. <https://doi.org/10.4236/ojg.2016.69078>
- Bui, D. T., Tsangaratos, P., Ngo, P. T. T., Pham, T. D., & Pham, B. T. (2019). Flash flood susceptibility modeling using an optimized fuzzy rule based feature selection technique and tree based ensemble methods. *Science of the Total Environment*, 668, 1038–1054. <https://doi.org/10.1016/j.scitotenv.2019.02.422>
- Callberg et al. (n.d.). OBFUSCATION TECHNIQUES FOR ENHANCING SOFTWARE SECURITY. United States Patent, 2003.
- Chen, K., Xie, Q., Qiu, W., Li, N. (Computer scientist), Tzeng, W.-G., Association for Computing Machinery. Special Interest Group on Security, A., ... *ACM Digital Library*. (2013). *ASIA CCS'13 : proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security : May 8-10, 2013, Hangzhou, China*.
- Chua, M., & Balachandran, V. (2018). Effectiveness of android obfuscation on evading anti-malware. *CODASPY 2018 - Proceedings of the 8th ACM Conference on Data and Application Security and Privacy*, 2018-January, 143–145. Association for Computing Machinery, Inc. <https://doi.org/10.1145/3176258.3176942>
- Dong, S., Li, M., Diao, W., Liu, X., Liu, J., Li, Z., ... Zhang, K. (2018). Understanding android obfuscation techniques: A large-scale investigation in the wild. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, LNICST, 254, 172–192. Springer Verlag. https://doi.org/10.1007/978-3-030-01701-9_10
- Duan, Q., Dong, P., Xi, L., & Ma, X. (2018). Signal prediction based on boosting and decision stump. In *Int. J. Computational Science and Engineering* (Vol. 16).
- Faruki, P., Hossein, F., Vijay, L., Mauro, C., & Manoj, G. (2017). Android Code Protection via Obfuscation Techniques: Past, Present and Future Directions. *ACM Computing Servwys*. <https://doi.org/10.1145/0000000.0000000>
- Lee, A., Taylor, P., & Kalpathy-Cramer, J. (2017, December 1). Machine Learning Has Arrived! *Ophthalmology*, Vol. 124, pp. 1726–1728. Elsevier Inc. <https://doi.org/10.1016/j.opthta.2017.08.046>

Li, J., Wang, Z. L., Zhao, H., Gravina, R., Fortino, G., Jiang, Y., & Tang, K. (2017). Networked human motion capture system based on quaternion navigation. *BodyNets International Conference on Body Area Networks*. <https://doi.org/10.1145/0000000.0000000>

Mahesh, B. (2018). Machine Learning Algorithms-A Review Machine Learning Algorithms-A Review View project Self Flowing Generator View project Batta Mahesh Independent Researcher Machine Learning Algorithms-A Review. *International Journal of Science and Research*. <https://doi.org/10.21275/ART20203995>

Maiorca, D., Ariu, D., Corona, I., Aresu, M., & Giacinto, G. (2015). Stealth attacks: An extended insight into the obfuscation effects on Android malware. *Computers and Security*, 51, 16–31. <https://doi.org/10.1016/j.cose.2015.02.007>

Mirzaei, O., de Fuentes, J. M., Tapiador, J., & Gonzalez-Manzano, L. (2019). ANDRODET: An adaptive Android obfuscation detector. *Future Generation Computer Systems*, 90, 240–261. <https://doi.org/10.1016/j.future.2018.07.066>

Park, M., You, G., Cho, S. J., Park, M., & Han, S. (2019). A framework for identifying obfuscation techniques applied to android apps using machine learning. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 10(4), 22–30. <https://doi.org/10.22667/JOWUA.2019.12.31.022>

Reva University, Institution of Electronics and Telecommunication Engineers (India), IEEE Communications Society. Bangalore Chapter, IEEE Circuits and Systems Society. Bangalore Chapter, Institute of Electrical and Electronics Engineers. Bangalore Section., & Institute of Electrical and Electronics Engineers. (n.d.). *Proceedings of 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAEECC-2018) : 9-10 February 2018, Reva University, Bengaluru, India.*

Ridzuan Daud, N., & Wolfe Corne, D. (n.d.). *HUMAN READABLE RULE INDUCTION IN MEDICAL DATA MINING: A SURVEY OF EXISTING ALGORITHMS.*

Schulz, P., & Plohm, D. (n.d.). *Communication and Distributed Systems Code Protection in Android.*

Tu, M. C., Shin, D., & Shin, D. (2009). A comparative study of medical data classification methods based on decision tree and bagging algorithms. *8th IEEE International Symposium on Dependable, Autonomic and Secure Computing, DASC 2009*, 183–187. <https://doi.org/10.1109/DASC.2009.40>

V, B. R., Reddy, P., P, H., & U Patil, M. (2015, July 23). DROIDSWAN: Detecting Malicious Android Applications Based on Static Feature Analysis. 163–178. *Academy and Industry Research Collaboration Center (AIRCC)*. <https://doi.org/10.5121/csit.2015.51315>

Wong, M. Y., & Lie, D. (n.d.). Tackling runtime-based obfuscation in Android with Tiro. Retrieved from <https://www.usenix.org/conference/usenixsecurity18/presentation/wong>

Zhang, X., Breiter, F., Luechinger, E., & O'Shaughnessy, S. (2021). Android application forensics: A survey of obfuscation, obfuscation detection and deobfuscation techniques and their impact on investigations. *Forensic Science International: Digital Investigation*, 39, 301285. <https://doi.org/10.1016/j.fsidi.2021.301285>

Zhu, Q., Sun, Z., Liang, X., Xiong, Y., & Zhang, L. (2020). OCoR: An Overlapping-Aware Code Retriever. *Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*, 883–894. *Institute of Electrical and Electronics Engineers Inc.* <https://doi.org/10.1145/1122445.1122456>

ÖZGEÇMİŞ

Adı Soyadı : Sadık SARIBIYIK
Doğum Yeri ve Tarihi : Osmaniye, 1991
Yabancı Dil : İngilizce, Almanca

Eğitim Durumu
Lise : Osmaniye Anadolu Lisesi
Lisans : Çukurova Üniversitesi Bilgisayar Müh.
Yüksek Lisans : Bursa Uludağ Üniversitesi

Çalıştığı Kurum/Kurumlar : Bursa Uludağ Üniversitesi- Bilgisayar Mühendisliği

İletişim (e-posta) : sadiks@uludag.edu.tr